UNIVERSITY OF BREMEN
AND
GERMAN AEROSPACE CENTER
INSTITUTE OF
ROBOTICS AND MECHATRONICS

The Robot Architecture Framework

A systematic approach for describing the architecture of complex, autonomous robotic systems.

Dipl. Ing. Andreas Dömel

Vollständiger Abdruck der vom Fachbereich 3 (Mathematik und Informatik) der Universität Bremen zur Erlangung des akademischen Grades eines

Doktors der Ingenieurswissenschaften (Dr.-Ing.)

genehmigten Dissertation.

1. Gutachter:	Prof. Dr. Michael Suppa *Universität Bremen*
2. Gutachter:	Prof. Dr. Christian Schlegel *Technische Hochschule Ulm*

Die Dissertation wurde am 18.12.2024 bei der Universität Bremen eingereicht.
Das Kolloquium fand am 9.4.2025 statt.

Abstract

Many autonomous robots are currently designed based on the system architect's individual experience and knowledge, taking into account given factors such as application and environment. The complexity of robots, but also the diversity of systems, applications and fields of use, make it difficult to compare these individual solutions and to transfer architectural approaches to new system designs.
This thesis answers the research question of how the knowledge about robot architecture contained in existing systems can be made accessible. The developed process enables the systematic capturing of the architecture of autonomous robot systems. The resulting structured architecture descriptions make it possible to identify similarities and differences in the architectures of different systems despite the complexity and heterogeneity of robots.

The methodology chosen is an extension of the Architecture Framework concept. This approach, which originates from the field of software development, is used to capture the architectures of distributed software systems. This work transfers and extends the approach to the domain of robotics by defining a Robot Architecture Framework. The architecture description process based on this framework enables the architecture of existing robotic systems to be identified.
The applicability of the process is demonstrated using three very different robotic systems. It is shown that compact architecture descriptions can be generated despite the complexity and heterogeneity of the systems. In addition, the broad applicability of the defined process is also conceptually validated.

Zusammenfassung

Viele autonome Roboter werden derzeit auf Grundlage der individuellen Erfahrung und des Wissens des Systemarchitekten unter Berücksichtigung gegebener Faktoren wie Anwendung und Umgebung entworfen. Die Komplexität von Robotern, aber auch die Vielfalt von Systemen, Anwendungen und Einsatzgebieten erschweren den Vergleich dieser individuellen Lösungen und die Übertragbarkeit von Architekturansätzen auf neue Systemdesigns.
In dieser Arbeit wird die Forschungsfrage beantwortet, wie aus bestehenden Systemen das darin enthaltene Wissen über die Roboterarchitektur zugänglich gemacht werden kann. Der entwickelte Prozess ermöglicht die systematische Erfassung der Architektur autonomer Robotersysteme. Die resultierenden strukturierten Architekturbeschreibungen erlauben es, trotz der Komplexität und Heterogenität von Robotern, Gemeinsamkeiten aber auch Unterschiede in den Architekturen der verschiedenen Systeme zu identifizieren.

Die gewählte Methodik ist eine Erweiterung des Architecture Framework Konzepts. Dieser aus dem Bereich der Softwareentwicklung stammende Ansatz dient der Erfassung von Architekturen verteilter Softwaresysteme. Diese Arbeit überträgt und erweitert den Ansatz auf die Domäne der Robotik durch die Definition eines Robot Architecture Frameworks. Der darauf aufbauende Architekturbeschreibungsprozess ermöglicht die Erfassung der Architektur existierender Robotersysteme.
Die Anwendbarkeit des Prozesses wird exemplarisch an drei sehr unterschiedlichen Robotersystemen demonstriert. Es wird gezeigt, dass trotz der Komplexität und Heterogenität der Systeme kompakte Architekturbeschreibungen erzeugt werden können. Darüber hinaus wird die breite Anwendbarkeit des definierten Prozesses auch konzeptionell nachgewiesen.

Contents

List of Figures

List of Tables

List of Definitions

Architecture Framework Elements

Chapter one

Introduction

Humanoid robots that can perform tasks in industry as well as in everyday life are a great dream of mankind. Several large companies and start-ups are currently working on the development of such robots. New videos showing the impressive progress of these systems are released on a regular basis. Despite the efforts being put into the development of these systems, there is still a long way to go before humanoid robots can be a real help. At the moment, considerable efforts have to be made in order to show individual demonstrations with these systems, where the tasks and environments are simplified in comparison to the real application. So far, it has not been possible to give humanoid robots the ability to master the complexity of their own hardware and software and thus to solve tasks in real applications.

Parallel to this development, humans have succeeded in developing machines that make their work easier. However, these are not modeled on humans. It was with the industrialization that changed the nature of work. Instead of unique handcrafted items, it became possible to manufacture standardized products in large quantities. The necessary work steps could be distributed to different people and gradually also to machines. This classic automation, in which each work step is carried out by a machine specialized in precisely this task, further increased productivity, but was usually only financially worthwhile for very large quantities. Even quantities of several hundred products per day, such as in large automobile factories, do not justify this expense and are therefore still heavily based on human labor today.

The early 1960s saw a new development, the first industrial robots. These machines can carry out any movement within their workspace, taking kinematic restrictions into account. The same machine could therefore be used for many different tasks, which

could then be programmed according to the application. Whether intentionally or implicitly dictated by the requirements, these machines often resemble an arm, which brings us closer to the idea of an artificial human being, at least from a mechanical point of view. The requirement for these robots is to achieve high positioning accuracy for a large payload at the highest possible speed. These high requirements from a mechanical point of view have led to control technology taking on a central role in robotics at an early stage. It is no longer just the stiffness of the mechanics that determines the positioning accuracy, but the superimposed control system can correct position deviations on the basis of sensor measurements. This closed feedback loop is the first step towards an autonomous robot, as the system is able to correct its position by itself. However, classic industrial robots are not considered autonomous robots, as the actual tasks, such as transporting or processing workpieces, must be specifically programmed. If a parameter such as the environment, the workpiece, etc. changes, the program must be revised or recreated by a human.

International competition is accelerating product cycles. In addition, digitalization allows for increased configuration and individualization of products. High quantities of identical products are becoming rarer as a result. In future, the industry will need more flexible production that meets these requirements. Initial approaches include the expansion of robot systems with additional sensors. This incremental integration leads to an increase in system complexity, which is no longer manageable in a highly flexible production. In practice, human workers are therefore still predominantly used for such tasks.
The old dream of mankind to build a machine that also resembles human labor in its flexibility is thus clearly gaining ground again. In addition, this development is reinforced by other factors such as the aging of society in industrialized countries or the exploration of space. The motivation to develop autonomous robotic systems with human-like flexibility for certain tasks is thus emerging from completely different areas.
However, there is currently no commercially available system that even comes close to meeting these requirements. Autonomous robots in real applications can only perform less complex and very specialized tasks so far. The flexibility of these systems is limited to being able to perform their tasks under different environmental conditions. For example, these systems can vacuum, clean homes or mow the lawn autonomously without the support of robotics experts. Autonomous driving of cars is also conceivable in the near future. The focus in the development of these systems is on being able to perform specific tasks reliably despite a partially unknown environment. More complex tasks, such as the targeted manipulation of the physical environment, cannot yet be solved by autonomous systems.

1.1 Problem Statement

The development of robots that are able to solve more complex tasks autonomously has therefore been a central research area in robotics for a long time. Various approaches are being pursued to achieve this goal. Particularly in the hardware sector, but also in parts of AI research, attempts are being made to use humans as a model. The idea behind this is that if you imitate humans closely enough, you can create a machine that is capable of performing human-like tasks. So far, this approach has failed, at least from a practical application perspective, due to the enormous complexity of humans.
Another approach is to consider the required functions separately. In robotics research, methods of mechanics, control engineering, perception, motion planning, etc. are being further developed for this purpose. Great progress has been made here in recent years, which has also been supported by the development of computing power. The great advantage of this approach is that the problem can be broken down into individual, manageable sub-problems. This separation of the problem also makes it easier to present, compare and document the solutions. Much of the scientific work in robotics therefore deals with subproblems and the development of methods for special problem classes. This approach is correct and important, as the overall problem cannot be solved without an existing solution for sub-problems. However, the reverse conclusion, that if the identified sub-problems are solved, the overall problem is solved, is wrong and there are several reasons for this, which are listed here in a very abbreviated form:

1. **Unknown subproblems:**
   Some subproblems are easy to identify and are also investigated in other areas such as classical automation technology. However, there is no approach to prove the completeness of this subdivision. It is therefore likely that important subtopics will not be identified. Small gaps between the various sub-areas in particular can be difficult to identify.

2. **Unrealistic or irrelevant goals, prerequisites and conditions:**
   In order to be able to consider a subproblem, assumptions must be made about the interfaces. This applies to the prerequisites and conditions as well as the goals to be achieved. For example, perception methods are often tested on data sets. These typically consist of images and ground truth positions of the objects depicted. However, the advantage of being able to test different methods in a standardized way bears the risk that methods are optimized or even

developed for the test. Other methods that may be better suited to the overall system and are more robust against sensor errors, occlusions and changing lighting conditions, for example, are not identified and further developed. The objectives of the sub-topics can also drift apart due to the separation. For example, in path planning, proving the probabilistic completeness of a planner is an academic goal. For the application of the planner on a robot, however, it is irrelevant whether the planner finds a solution after an infinitely long time or not.

3. **Dependence between components:**
   Even if the conditions have been chosen realistically, this does not mean that the combination of components will work. Often, trade-offs have to be made between different goals, e.g. accuracy vs. speed. Since one component often provides the input for another component, there are dependencies that must be considered in their entirety.

4. **Non-disjoint problems:**
   Some sub-problems overlap with others and therefore cannot be considered separately. For example, viewpoint planning depends on perception, kinematics and the environment. These sub-problems become more complex the more different sub-problems have to be integrated or combined.

The conceptual analysis of the overall system, i.e. the robot architecture, is therefore a central aspect of research in the field of autonomous systems. The first approaches to this are the classic robot architectures, which form the basis for early autonomous systems such as Shakey the Robot [90]. Later, different approaches were combined in various architectures in so-called layers, e.g. 3T architecture [13]. To counter the increasing complexity of the software of autonomous systems, approaches from software engineering were also increasingly used in robotics. For example, software frameworks support the development of modern robot systems by providing tools and structural standards. Model-based approaches separate the design and implementation of robot software and robotics ecosystems are intended to enable different stakeholders to contribute to robot solutions without solving the overall problem. In summary, however, it can be stated that there is currently no generally accepted standard architecture for autonomous robotic systems. The most widespread are currently software frameworks such as ROS, which explicitly leave the majority of architectural decisions to the respective developer.

There are many reasons for this, including the complexity and diversity of robot systems. In addition, robots are used in a wide variety of environments for a wide variety of tasks. A Mars rover has different requirements in terms of reliability and

speed than a service robot in a restaurant. It is therefore obvious that the architecture of the two systems should also differ.

Autonomous robot systems are therefore developed and built individually for the respective application. The current state of the art can be summarized well with a quote from the current edition of the Springer Handbook of Robotics:

> *"Designing a robot architecture is much more of an art than a science ...the decisions made by a developer of a robot architecture are influenced by their own prior experiences, their robot and its environment, and the tasks that need to be performed."* — *Kortenkamp, Simmons, and Brugali [62]*

The development and design of an autonomous robot is essentially based on the developer's experience. This makes it possible to build a system that fulfills its tasks under the given conditions. For example, there are robots that clean [72], help around the house [61], make coffee [41] or assist with construction work [14].
These impressive demonstrations push the boundaries of what is possible and show that robots are, in principle, capable of solving these tasks. All of these systems are based on the latest technologies in the various subcategories (perception, knowledge representation, control technology, etc.). The individual components of these systems are partly described in scientific publications and can be used, compared and further developed by others.

However, knowledge about the architecture of these systems is difficult to access. Videos and demonstrations of the systems show the results achieved. Yet the system architecture and its contribution to the result are difficult to deduce. Publications provide a deeper insight into the system, whereby the focus here is on new contributions to the state of research. A complete description of the architecture cannot be found in publications either. The software of some systems is completely open source. Insight here is naturally limited to the software components and their architecture. In addition, these software frameworks are often very extensive. Basic concepts that are transferable to other systems are therefore difficult to identify. A more detailed look at related work can be found in Chapter 2.
In summary, it can be said that it is currently difficult to learn about architecture from existing robotic systems due to the lack of accessible information. This applies both to the development of new systems and to the evolution of existing systems. This leads also to a lack of comparability of system concepts.

1.2 Contributions

The main contribution of this work is a process for describing the architecture of robots in a structured way, in order to make the architectural knowledge contained in existing robotic systems accessible. These explicit architecture descriptions serve several purposes:

When developing a new system, it is possible to consider which aspects have been taken into account by other systems. This helps to consider all important aspects already in the design phase. If the requirements are compatible, existing concepts can be adopted, adapted or extended. If no solutions are available in existing systems, it becomes explicitly clear for which aspects new concepts have to be developed.

Explicit architecture descriptions are also useful for existing systems. On the one hand, new findings from other systems can be integrated, provided they are suitable for the respective robot. But the architecture description is also helpful for the evolution of a system. For example, if the tasks of a robot change, the architecture description can be used to systematically check which concepts are still valid and where conceptual changes are necessary.

In addition to the development of robot systems, architecture descriptions also help to compare systems at a conceptual level. The explicit description allows to identify and compare different approaches to similar problems. Conversely, the general applicability of sub-concepts across very different systems can also be demonstrated. The structured description also makes it possible to focus on relevant aspects of the system solution without having to consider the entire system in each case.

To achieve this, the following requirements apply to the architecture description:

1. It must be compact enough that complex systems can be described.
2. It must be possible to describe very different systems.
3. It must be possible to describe the system in sub-aspects.

This work therefore aims to develop a process that makes it possible to capture the architectures of complex robots in such a way that the resulting descriptions fulfill these requirements.

To achieve this goal, several individual steps were taken. First, a suitable methodology for the process definition was identified. This is based on the Architecture Framework approach, which was developed for distributed software systems and standardized in ISO 42010. In a second step, this approach was transferred to the field of

robotics by defining a *Robot Architecture Framework* (*RoAF*). With the help of this framework, a process was then developed that can be applied to a wide variety of robot systems and makes it possible to generate structured architecture descriptions of these robots. Finally, the process is verified both exemplarily on the basis of three created architecture descriptions and conceptually on the basis of the methodology used with regard to the fulfillment of the requirements. In addition, validation is carried out with the goal of making the architectural knowledge contained in the systems utilizable.

1.3 Thesis Outline

The structure of the work is based on the described approach and is organized into the following chapters.

Chapter 1: Introduction contains an introduction to the topic, describes the problem and the contributions of this work

Chapter 2: Related Work gives an overview of various works that deal with the topic of the architecture of robotic systems.

Chapter 3: Foundations of the Robot Architecture Framework presents the methodological and conceptual foundations of this work. This is the concept of the architecture framework on the one hand and the explicit definition of the robotics domain on the other.

Chapter 4: The Robot Architecture Framework describes the full *RoAF* and its components. These are the domain-specific stakeholders and concerns as well as the architecture viewpoints and architecture model abstraction types used.

Chapter 5: Robot Architecture Description Process defines the architecture description process based on the *RoAF*.

Chapter 6: Verification and Validation verifies the *RoAF* with regard to the identified requirements for architecture descriptions and validates it with regard to the goal of making knowledge from existing robot systems accessible.

Chapter 7: Discussion & Future Work discusses the decisions made for the *RoAF*. In addition, possible next steps for the further development of the *RoAF* but also for the general topic of robot architecture are given.

Appendix: Architecture Descriptions contains the architecture descriptions created for the evaluation of the *RoAF*. These are a complete architecture description of the industrial, mobile manipulator AIMM, the mission view of the exploration rover LRU2, and the hardware view of the drone ARDEA.

Glossary: All technical terms used in this work are listed and defined in the glossary. In the text, the technical terms are in italics and in Camel Case and provided with a link to the corresponding glossary entry.

Chapter two

Related Work

The systematic description of architectural concepts of existing systems has so far received little attention in robotics. However, the topic of robotics architecture itself has been discussed in robotics from the very beginning. The focus is on how architectural concepts can be used to develop systems. In contrast, *RoAF* defines a process for deriving architectural concepts from existing systems. Despite this opposed approach, the connecting element is the architectural concept itself. Hence, this work can be used to identify relevant aspects for robot architectures, as these form the basis for the description process. Therefore, this chapter provides an overview of the various works on robot architectures. Firstly, the classic robot architectures are presented, each of which defines how the components of a robot should interact. Then, different types of software frameworks are presented that specify robotic architectural decisions to varying degrees.

In addition to the works that explicitly deal with robot architectures, there are a large number of publications that describe the systems themselves. These often also contain sections that describe the architecture of the respective system. For a variety of reasons, the approach of creating architecture descriptions based on these publications was not chosen for the *RoAF*. Therefore, in this chapter the problems of extracting architectural knowledge from system publications are discussed using various systems and their publications.

Since no suitable methodology for describing the architectures of existing systems could be found in robotics, an approach from another domain was chosen. This is the concept of the architecture framework, described in ISO 42010, which was developed

for distributed software systems. One of the advantages of this concept is that it has already been applied to a variety of domains. In this thesis, the concept is applied to the domain of robotics in order to develop a systematic architecture description process. This chapter therefore gives some examples of how system architectures outside robotics are described using architecture frameworks.

2.1 Classical Architectures of Autonomous Robots

In the early days of the development of autonomous robotic systems, the architecture of the system was an important topic and the subject of numerous publications and discussions. The architecture issue was closely linked to the question of how artificial systems could achieve intelligent behavior. The classical approach according to the sense-reason-act paradigm was used, for example, in Shakey the Robot [90] depicted in Figure 2.1. The robot architecture consists of dividing the system into a module for perceiving the environment, a module for planning behavior and a module for executing actions (Nilsson [89]). The perception module integrates information into a world model. Based on this world model, a logical planner such as STRIPS [42] creates a plan. This plan is then executed by the action module. With this approach, systems can perform tasks in a targeted manner, but this approach also has some weaknesses and open questions that have not been answered to date, e.g. there is the problem of symbol grounding (Harnad [52]), incomplete knowledge is difficult to represent and dynamic environments lead to permanent replanning.

To address some of these problems, Brooks developed the subsumption architecture [17]. Sensor data is used directly to trigger actions. By superimposing different modules of different abstraction levels, which are executed in parallel, systems can also solve tasks, such as navigation in an office environment. The intelligent behavior of the system is achieved without an abstract representation of the environment or the capabilities [19]. A planning component is also not required [18]. The architecture thus solves some of the problems of the classical approach, e.g. the symbol grounding problem. However, the architecture shows weaknesses when it comes to solving more complex tasks such as manipulation tasks. The behavior becomes unmanageable due to the lack of subdivisibility (Hartley and Pipitone [53]).

This complementarity of the two architectures formed the basis for research in the 1990s. Numerous architectures were presented that attempt to combine the advan-

[1] source: https://images.computerhistory.org/revonline/images/x279.83p-03-01.jpg?w=600

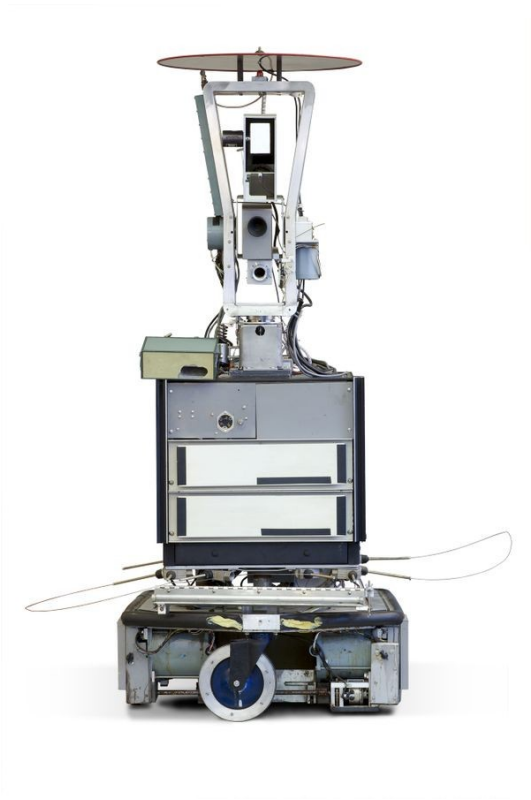

Figure 2.1: *Shakey[1] was one of the first autonomous robotic systems developed by the Stanford Research Institute in the late 1960s (Nilsson [89]). It already consisted of many components that still define complex autonomous systems today. The robot had a mobile base, various sensors and interfaces that enabled text-based task assignment. The interaction of various software components such as image processing, logic planner and world model enabled the robot to move boxes in an office environment in a targeted manner. The resulting complexity was addressed by a first robot architecture based on the sense-plan-act paradigm.*

tages of the respective approaches. To achieve this, the system is often divided into different layers. The lower layers implement the reactive behavior, the upper layers the planned actions. For example, the Atlantis architecture ([47],[45]) introduces three layers: controller, sequencer and deliberator. The SSS architecture [29] introduces three layers that divide the system behavior into the Servo, Subsumption and Symbolic levels. The Task Control Architecture [118] divides the tasks into hierarchical subtasks, which are also executed in parallel, and thus achieves a connection between the symbolic level and the reactive behavior. Other architectures divide the system into two layers, e.g. Firby and Slack [43] into the planning level of the Reactive Action Packages and an underlying level of the Reactive Skill Networks. Probably the best-known representative of these architectures is the 3T architecture [13], which defines the Reactive Skills, Sequencing and Deliberation levels. The

LAAS architecture [3] also belongs to this category of architectures and divides the system into a functional level, an execution control level and a decision level. A good overview of these different architectures can be found in Gat [46].

At the beginning of the 2000s, there was a clear change of direction in the development of robot architectures. This is well described in the publication "Architecture, the backbone of robotic systems" [30]. On the one hand, it describes the great importance of architecture for the development of autonomous systems. On the other hand, it also highlights the difficulties involved in architecture development. From the large number of architectures presented, it becomes clear that there is no one single correct architecture, but that different combinations of concepts are required depending on the system and application. Coste-Maniere and Simmons [30] propose a framework that makes it possible to combine different concepts. At the same time, the complexity of robotic systems has become increasingly important and expands the requirements to architectures to include the ability to master software complexity.

A later representative of these architectures is the CLARAty architecture [139], which is also one of the first robotic software frameworks. Purely architectural publications have been very sparse in the last two decades. Contributions on robotics architecture are mostly integrated into software frameworks or system papers.
More recent architectural descriptions are the Interaction-oriented Cognitive Architecture [92],[93], which is aimed at systems with human-robot interaction. A derivative of CLARAty is the Intelligent Robot System Architecture (IRSA) [5], which has been implemented on some NASA JPL robotic systems, e.g. by Karumanchi et al. [60]. Another example is the FLEXHRC+ architecture [33], which is suitable for human-robot collaboration, e.g. in furniture assembly. All of these architectural descriptions are individual solutions in a specific context. Comparability or even approaches for systematically developing architectures are not yet available. The current state of research is summarized by Kortenkamp, Simmons, and Brugali [62] as follows:

> *"Designing a robot architecture is much more of an art than a science ...the decisions made by a developer of a robot architecture are influenced by their own prior experiences, their robot and its environment, and the tasks that need to be performed."* — *Kortenkamp, Simmons, and Brugali [62]*

Based on the findings of classical architectures, the *RoAF* does not define the universal architecture for autonomous robots, but enables the description and comparison of different architectures and their respective contexts.

2.2 Robotic Software Frameworks

Classical architectures define which components are used to implement robotic behaviors. In addition, there are usually structures such as layers for different aspects, e.g. a deliberative layer for symbolic representations. This defines the architecture of a robot in an abstract way. Technical aspects such as how to implement communication between modules are not considered.

Over the decades, autonomous robot systems have become increasingly complex. This applies to the hardware, which has evolved from simple mobile robots to walking humanoids. Above all, however, the robotics software has developed from simple systems with a manageable number of interconnected components to distributed software systems with often more than 100 flexibly interacting modules, which poses new challenges for robotics.

The complexity of these systems must be mastered both in operation and during development, modification or expansion. Individual components and partial solutions should be reusable and solutions from others should be easy to integrate. Furthermore, autonomous robots today are the result of teamwork. It must therefore be possible to distribute development among several people.

Software frameworks are used in software development for this purpose. They enable the implementation of specific applications, but provide a framework in the form of design patterns or structures. This facilitates the exchange of individual system parts and enables the definition of roles with responsibilities. This led to the introduction of software frameworks for developing robot software in the early 2000s.

Unfortunately, individual terms such as “framework” and “architecture” are in some cases used for different things in the literature. According to ISO 42010, an *Architecture Framework* is the methodology for harmonizing different *Architecture Descriptions* of systems in a domain. A software framework, on the other hand, is used to make software development more efficient by providing structures, tools or even partial solutions. These are initially two very different things that only share the term framework.

This work applies the concept of the *Architecture Framework* to the domain of robotics and thus defines a *Robot Architecture Framework*. Software frameworks that have been developed for the development of software for robot systems are accordingly referred to as robot software frameworks. One framework is used to describe robot architectures, the other to develop software for robots. However, software frameworks usually define structures that determine the software architecture of

the software developed with them, for example to enable the interchangeability of components.

A software framework that is used on a robot thus defines the software architecture of the robot to a certain extent. This extent differs significantly between the various robotic software frameworks and is used in this section to categorize the software frameworks into three different categories.

If the architecture refers to the technical implementation of the software on the system, such as communication patterns or package management of the individual components, then this is largely orthogonal to classic robot architectures. In the following, these software frameworks are presented as architecture-agnostic software frameworks.

Other software frameworks define in detail which components exist on the system and how they are linked. In doing so, they define to a large extent the aspects addressed by the classical architectures, partly explicitly but often also implicitly through the implementation of the software framework itself. In the following, these software frameworks will be referred to as robot architecture software frameworks.

The third category of software frameworks are the model-based software frameworks. These separate the conception of a system and the implementation itself. The robot architecture is therefore not fully defined by the software framework but can differ from system to system. However, the conception is governed by rules and structures and thus defines a part of each system architecture.

In the following, various works from each of these three software framework categories are presented.

2.2.1 Architecture-agnostic software frameworks

This category of robotics software frameworks focuses strongly on the interchangeability of components and the cross-institutional development of robotics software solutions. Here, the framework primarily serves to ensure interchangeability and compatibility between different solutions and thus increases the reusability of solutions. The robot architecture itself takes a secondary role.

The idea of OROCOS [25] was to create an open source platform that enables software components to be integrated into the framework in a reusable manner. This should enable various stakeholders, such as industry and research, to make their contributions available. These modules are then combined in components. Instead of

specifying a robotic architecture, the idea was to create patterns similar to software patterns that support the design of a system.
YARA [28] is another example of a software framework that hardly specifies a robotic architecture, but instead focuses on the interchangeability of components and a communication solution.
The best-known and most widespread representatives of these software frameworks are ROS [32], [31] and ROS2 [76]. ROS is primarily a middleware and does not define much architectural specifications. This means that ROS components can be easily integrated into systems. However, reusability is limited to individual modules. Even at the middleware level, ROS itself defines only few architectural concepts. For this reason, Malavolta et al. [77] has examined a large number of ROS software repositories and presented guidelines for the development of ROS-based systems.
Non-robotics-specific software frameworks that focus on the middleware of a system, e.g. based on the DDS or OPC-UA standards, are also used in robotics.

2.2.2 Robot architecture software frameworks

The fundamental idea of this category is that software frameworks can be used in robotics not only to solve software engineering problems, but also to apply constraints of a robot architecture to systems. This variant of software frameworks therefore defines the robot architecture of the respective target system. CLARAty [139] is an early representative of these software frameworks and at the same time a late representative of classical robotics architectures. The framework allowed to use the CLARIty architecture on various rovers (Nesnas et al. [88]).
The “Software Architecture framework for service robots (SAFSR)” [73] is used in the ASORO lab for various service robots with different tasks. The framework divides the software into different layers (Device, Modality, Execution & Control, Cognitive). In addition, various modules are defined within the layers, e.g. Attention Directed Dialog Modules.
Another robotic architecture software framework consisting of KnowRob [127], [6] and CRAM [7] is used on the robots of the Institute for Artificial Intelligence at the University of Bremen. One focus of these frameworks is on ontology-based knowledge representation. The main area of application of the frameworks are robots that are intended to perform household tasks.
A framework for autonomous robot systems in an industrial context is SkiRos [97] and the subsequent development SkiRos2 [80]. This framework also uses an ontology-based knowledge representation. SkiRos2 has been used on various robot systems in

an industrial context that perform logistics tasks and simple assembly tasks.
The ARMAR-X framework [136] developed and used at KIT's H2T also defines the architecture in layers (middleware, robot framework, application). This framework is also used on various robots at the H2T lab. The focus here is on humanoid robots. An important component of the robot architecture software framework is the memory of the robot MemoryX [91].
The common characteristic of all these frameworks is that they are very complex software systems. Therefore, these types of frameworks, even if they are open source, are mainly used by their developers. A broad community or even a widespread standard solution has not been established yet. However, due to the complexity of these systems and the close integration of architecture and implementation, it is very difficult to evaluate, compare or learn from the influence of the architecture solution of the respective framework separately. In industry, the approach of robot architecture software frameworks, although for less complex systems, is already being used by various providers, e.g. Artimind[2], Instrinsic[3], to offer more efficient solutions for a wide range of robot systems.

2.2.3 Model-based software frameworks

Another category of robotics software frameworks are model-based approaches. Model-based approaches also focus on the interchangeability and reusability of components or entire partial solutions. However, this is not achieved directly through the implementation of the software framework, but through an implementation-independent modeling of the system. The implementation is then carried out on the basis of the model and is supported by the tools of the framework.

The SmartSoft project [103] implements this approach by developing the software on the basis of models rather than code. To create the software for a system, models are first created at various levels of abstraction. The components and their connections are derived from these models. The code is then created or generated from this description. All these steps are supported by a software tool chain. Thanks to implementation-independent modeling, devices and middleware can be connected and exchanged at a later stage. In order to ensure the combinability of the various components, structures and thus the robot architecture are partially predefined compared to the architecture agnostic frameworks (Lutz et al. [74]).

[2]https://www.artiminds.com/
[3]https://www.intrinsic.ai/

RobMoSys is continuing to develop these ideas with the aim of creating a software ecosystem that can be used to put together individual system solutions for different areas of application (Schlegel et al. [102]). The different roles make it possible to contribute the respective expertise without having to know the overall solution. The robot architecture is defined in so-called architectural patterns.

Another example of a model-based software framework was developed in the BRICS project [8]. The aim of the project was to formalize the development process of robot systems and to support it with software tools. The BRICS Component Model [26] was developed for this purpose. The robot's software is modeled using models of different levels of abstraction. The software itself is then generated using the BRIDE software tools. The component model itself was extended in later work by composition patterns (Vanthienen, Klotzbuecher, and Bruyninckx [137]). With these, it is also possible to model the interaction of components.

More recent examples of model-based approaches are Architecture Description Languages, e.g. Adam et al. [1], Monthe, Nana, and Kouamou [81].

2.2.4 Robotic Software Frameworks in general

The studies by Brugali et al. [22] and Ahmad and Babar [2] provide an overview of the numerous robotic software frameworks. However, all robotic software frameworks have some limitations from the perspective of a robot architect.

Software focus Software frameworks have a strong focus on the robot's software. However, the software and its architecture are only one part of a robotic system. Important architectural decisions are therefore not covered by the software frameworks.

General applicability The goal of most software frameworks is to cover a wide range of applications, i.e. to provide solutions for different domains and robotic systems. Since the architecture is defined at the framework level, individual adaptations of the architecture to system requirements are difficult. Therefore, the architecture of software frameworks must be designed more generically than for individual systems, and system-specific design decisions cannot be represented in the software framework itself.

Unclear limitations Robotic software frameworks have limitations. Every structure comes with advantages and disadvantages. However, these considerations are

usually not documented for software frameworks. The applicability of the vast majority of software frameworks is also only proven by a relatively small number of systems.

Incompleteness Software frameworks do not completely define the architecture of a system. There is currently no software framework that allows a complete robot and its architecture to be designed. Some gaps, e.g. the missing hardware aspect, are obvious. Other aspects e.g. resource requirements or real-time capability are less obvious. Although some software frameworks consider these aspects, others do not. As the gaps are not documented it is difficult to categorize a software framework.

In summary, it can be said that software frameworks have made today's complexity of robot systems and thus progress in applied robotics possible. However, a standard solution for the development of robot systems has not yet emerged. Due to the complexity, it is also difficult to assess the contribution of different aspects of a framework to the system solution. The most widely used software framework in research, ROS, specifies very few architectural decisions. The success of this framework therefore lies more in its ease of use, a large community, useful tools and a high degree of freedom for the respective system developer. However, it is hardly possible to derive any insights into what makes a good system architecture.

RoAF focuses on describing the architecture of existing systems. The development of architectures and robots based on them is not the subject of this work. For this reason the *RoAF* itself is not a software framework but contains aspects which are addressed also by robotic software frameworks. Furthermore, the *RoAF* uses many insights from the field of software engineering, from the concept of the *Architecture Framework* to the various *Software Concerns*.

2.3 Robotic System Description

Another way to learn about the architecture of robots is to analyze existing robots. According to ISO 42010, every realized system has an architecture. By looking at current systems, it should therefore also be possible to derive the current state of research on robot architectures. This has the advantage that it becomes clear for which systems and applications the architecture was designed. In addition, the completeness of the architecture is inherently given for a functioning overall system.

Title	Category	Reference
Integration and Assessment of Multiple Mobile Manipulators in a Real-World Industrial Production Facility	system	Bogh et al. [12]
Autonomous pick and place operations in industrial production	system	Dömel et al. [34]
Toward fully autonomous mobile manipulation for industrial environments	system	Dömel et al. [35]
Sequential scene parsing using range and intensity information	method	Brucker et al. [21]
Combining object modeling and recognition for active scene exploration	method	Kriegel et al. [63]
RAFCON: A graphical tool for engineering complex, robotic tasks	method	Brunner et al. [23]
Efficient next-best-scan planning for autonomous 3D surface reconstruction of unknown objects	method	Kriegel et al. [64]
Repetition sampling for efficiently planning similar constrained manipulation tasks	method	Lehner and Albu-Schäffer [67]
Experience-based optimization of robotic perception	method	Durner et al. [38]
Automated Benchmarks and Optimization of Perception Tasks	method	Durner et al. [39]
Implicit 3D Orientation Learning for 6D Object Detection from RGB Images	method	Sundermeyer et al. [123]
The Repetition Roadmap for Repetitive Constrained Motion Planning	method	Lehner and Albu-Schäffer [68]
Autonomous Parallelization of Resource-Aware Robotic Task Nodes	method	Brunner et al. [24]
Visual Repetition Sampling for Robot Manipulation Planning	method	Puang et al. [95]
6DoF Pose Estimation for Industrial Manipulation Based on Synthetic Data	method	Brucker et al. [20]
Augmented Autoencoders: Implicit 3D Orientation Learning for 6D Object Detection	method	Sundermeyer et al. [124]
Unknown Object Segmentation from Stereo Images	method	Durner et al. [37]
Hybrid Planning System for In-Space Robotic Assembly of Telescopes using Segmented Mirror Tiles	method	Martínez-Moritz et al. [78]
Kinematic Transfer Learning of Sampling Distributions for Manipulator Motion Planning	method	Lehner, Roa, and Albu-Schäffer [70]
Robotic world models—conceptualization, review, and engineering best practices	method	Sakagami et al. [101]
CollisionGP: Gaussian Process-Based Collision Checking for Robot Motion Planning	method	Muñoz et al. [87]
Task-Level Programming by Demonstration for Mobile Robotic Manipulators through Human Demonstrations based on Semantic Skill Recognition	method	Mayershofer et al. [79]

Table 2.1: *Publications related to the AIMM system*

Robot systems are therefore potentially a very valuable source of advances in the field of robot architecture. The main problem, however, is that this knowledge is difficult to access. The reasons are elaborated in the rest of this section.
Robotic systems are mainly known to the outside world through their application. For almost every robotic system, there are videos showing how tasks are solved by the robot. But these demonstrations are not sufficient for more detailed insights into the system. Therefore, at least for the systems that come from the research area, there are one or more system papers for every more complex robotic system. In addition, there are usually a large number of method papers that use the system itself to evaluate a method or component.
Since a complete overview of system papers and, in particular, method papers is not feasible, the publications of the systems considered in this work, AIMM, LRU2 and ARDEA, are presented as examples:

Table 2.1 lists the publications of the AIMM system. A total of 22 system related publications were found. Of these, 3 publications focus on the system and its application. 19 publications focus on individual components or methods. For the LRU2 system, see Table 2.2, 40 publications were identified. Of these, 12 concern the system and its application. Individual components or methods are presented in 28 publications. As shown in Table 2.3, the ARDEA drone can be found in 31 publications, 9 of which focus on the system and 22 describe methods and individual components.

Title	Category	Reference
Stereo-vision based obstacle mapping for indoor/outdoor SLAM	system	Brand et al. [15]
LRU – Lightweight Rover Unit	system	Wedler et al. [143]
The LRU Rover for Autonomous Planetary Exploration and Its Success in the SpaceBotCamp Challenge	system	Schuster et al. [113]
First Results of the ROBEX Analogue Mission Campaign: Robotic Deployment of Seismic Networks for Future Lunar Missions	system	Wedler et al. [145]
From single autonomous robots toward cooperative robotic interactions for future planetary exploration missions	system	Wedler et al. [146]
Mobile manipulation for planetary exploration	system	Lehner et al. [69]
Towards Heterogeneous Robotic Teams for Collaborative Scientific Sampling in Lunar and Planetary Environments	system	Schuster et al. [114]
German Aerospace Center's advanced robotic technologyfor future lunar scientific missions	system	Wedler et al. [144]
Preliminary Results for the Multi-Robot, Multi-Partner, Multi-Mission, Planetary Exploration Analogue Campaign on Mount	system	Wedler et al. [141]
Finally! Insights into the ARCHES Lunar Planetary Exploration Analogue Campaign on Etna in summer 2022	system	Wedler et al. [142]
Mobile Manipulation of a Laser-induced Breakdown Spectrometer for Planetary Exploration	system	Lehner et al. [71]
Enabling Distributed Low Radio Frequency Arrays - Results of an Analog Campaign on Mt. Etna	system	Staudinger et al. [120]
ROBEX – COMPONENTS AND METHODS FOR THE PLANETARY EXPLORATION DEMONSTRATION MISSION	method	Wedler et al. [140]
Reachability and Dexterity: Analysis and Applications for Space Robotics	method	Porges et al. [94]
Multi-robot 6D graph SLAM connecting decoupled local reference filters	method	Schuster et al. [113]
Submap matching for stereo-vision based indoor/outdoor SLAM	method	Brand et al. [16]
RAFCON: A graphical tool for engineering complex, robotic tasks	method	Brunner et al. [23]
Software-in-the-Loop Simulation of a Planetary Rover	method	Hellerer, Schuster, and Lichtenheldt [55]
Experience-based optimization of robotic perception	method	Durner et al. [38]
Exploration with active loop closing: A trade-off between exploration efficiency and map quality	method	Lehner et al. [66]
Datasets of Long Range Navigation Experiments in a Moon Analogue Environment on Mount Etna	method	Vayugundla et al. [138]
Slip Modeling and Estimation for a Planetary Exploration Rover: Experimental Results from Mt. Etna	method	Bussmann et al. [27]
Distributed stereo vision-based 6D localization and mapping for multi-robot teams	method	Schuster et al. [116]
Relocalization With Submaps: Multi-Session Mapping for Planetary Rovers Equipped With Stereo Cameras	method	Giubilato et al. [50]
Rock Instance Segmentation from Synthetic Images for Planetary Exploration Missions	method	Boerdijk et al. [11]
A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments	method	Müller et al. [86]
Design and Implementation of a Modular Mechatronics Infrastructure for Robotic Planetary Exploration Assets	method	Fonseca Prince et al. [44]
Multi-Modal Loop Closing in Unstructured Planetary Environments with Visually Enriched Submaps	method	Giubilato et al. [51]
Towards Robust Perception of Unknown Objects in the Wild	method	Boerdijk et al. [9]
URSim - A Versatile Robot Simulator for Extra-Terrestrial Exploration	method	Sewtz et al. [117]
Interactive OAISYS: A photorealistic terrain simulation for robotics research	method	Müller et al. [86]
GPGM-SLAM: a Robust SLAM System for Unstructured Planetary Environments with Gaussian Process Gradient Maps	method	Giubilato et al. [49]
ROSMC: A High-Level Mission Operation Framework for Heterogeneous Robotic Teams	method	Sakagami et al. [99]
Robotic world models—conceptualization, review, and engineering best practices	method	Sakagami et al. [101]
Terrain-aware communication coverage prediction for cooperative networked robots in unstructured environments	method	Staudinger et al. [119]
Autonomous Rock Instance Segmentation for Extra-Terrestrial Robotic Missions	method	Durner et al. [36]
ReSyRIS - A Real-Synthetic Rock Instance Segmentation Dataset for Training and Benchmarking	method	Boerdijk et al. [10]
Uncertainty Estimation for Planetary Robotic Terrain Segmentation	method	Müller et al. [84]
A Laser-Induced Breakdown Spectroscopy (LIBS) Instrument for In-Situ Exploration with the DLR Lightweight Rover Unit (LRU)	method	Schröder et al. [111]

Table 2.2: *Publications related to the LRU2 system*

Title	Category	Reference
Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue	system	Tomic et al. [129]
Towards Autonomous MAV Exploration in Cluttered Indoor and Outdoor Environments	system	Schmid, Suppa, and Burschka [108]
From single autonomous robots toward cooperative robotic interactions for future planetary exploration missions	system	Wedler et al. [146]
Integration of an Automated Valet Parking Service into an Internet of Things Platform	system	Tcheumadjeu et al. [126]
Towards Heterogeneous Robotic Teams for Collaborative Scientific Sampling in Lunar and Planetary Environments	system	Schuster et al. [114]
ARDEA — An MAV with skills for future planetary missions	system	Lutz et al. [75]
The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration	system	Schuster et al. [115]
Preliminary Results for the Multi-Robot, Multi-Partner, Multi-Mission, Planetary Exploration Analogue Campaign on Mount Etna	system	Wedler et al. [141]
Finally! Insights into the ARCHES Lunar Planetary Exploration Analogue Campaign on Etna in summer 2022	system	Wedler et al. [142]
View Planning for Multi-View Stereo 3D Reconstruction Using an Autonomous Multicopter	method	Schmid et al. [104]
State estimation for highly dynamic flying systems using key frame odometry with varying time delays	method	Schmid et al. [107]
Stereo vision based indoor/outdoor navigation for flying robots	method	Schmid et al. [109]
Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation	method	Schmid et al. [105]
A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots	method	Tomic and Haddadin [128]
Learning quadrotor maneuvers from optimal control and generalizing in real-time	method	Tomić, Maier, and Haddadin [133]
Evaluation of acceleration-based disturbance observation for multicopter control	method	Tomić [130]
Local reference filter for life-long vision aided inertial navigation	method	Schmid, Ruess, and Burschka [106]
Simultaneous estimation of aerodynamic and contact forces in flying robots: Applications to metric wind estimation and collision detection	method	Tomić and Haddadin [131]
The flying anemometer: Unified estimation of wind velocity from aerodynamic power and wrenches	method	Tomić et al. [135]
External Wrench Estimation, Collision Detection, and Reflex Reaction for Flying Robots	method	Tomić, Ott, and Haddadin [134]
Robust Visual-Inertial State Estimation with Multiple Odometries and Efficient Mapping on an MAV with Ultra-Wide FOV Stereo Vision	method	Müller et al. [83]
Distributed stereo vision-based 6D localization and mapping for multi-robot teams	method	Schuster et al. [116]
Simultaneous contact and aerodynamic force estimation (s-CAFE) for aerial robots	method	Tomić et al. [132]
Efficient Terrain Following for a Micro Aerial Vehicle with Ultra-Wide Stereo Cameras	method	Müller et al. [85]
A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments	method	Müller et al. [82]
URSim - A Versatile Robot Simulator for Extra-Terrestrial Exploration	method	Sewtz et al. [117]
Interactive OAISYS: A photorealistic terrain simulation for robotics research	method	Müller et al. [86]
ROSMC: A High-Level Mission Operation Framework for Heterogeneous Robotic Teams	method	Sakagami et al. [99]
Robotic world models—conceptualization, review, and engineering best practices	method	Sakagami et al. [101]
Terrain-aware communication coverage prediction for cooperative networked robots in unstructured environments	method	Staudinger et al. [119]
Uncertainty Estimation for Planetary Robotic Terrain Segmentation	method	Müller et al. [84]

Table 2.3: *Publications related to the ARDEA system*

Even if the analysis can only be carried out as an example, some problems can be identified in extracting the architecture of robotic systems from the publications.

Amount of publications As shown as an example, there are a large number of publications on large robotic systems. These contain widely varying amounts of information regarding the architecture of the system. The effort involved in finding and reviewing all these publications, extracting the relevant information from the various publications and deriving an architecture from it is enormous. And for many systems in particular, this is practically impossible.

Extent of the individual publication While the overall literature on a system has a very large extent, the individual publications are limited by the specifications of the conferences and journals. This limited extent makes it difficult to present the entire architecture of a system in one publication. Ultimately, therefore, the overall system is usually presented at a very abstract level in order to then describe individual aspects of the system. Each individual publication provides therefore only an incomplete description of the system. Therefore, the architecture of the system cannot be derived from a single publication.

Publication period Complex robotic systems are often used and evolved over many years. Accordingly, the publications are distributed over a longer period of time. However, it is very difficult for external parties to recognize which information from the publications reflects the current state of the system and which information is outdated because newer concepts have replaced it. Deriving the current architecture of a robot from a collection of literature spanning 10 or more years is therefore very complex.

Publication focus Scientific publications must provide new insights that go beyond what is currently known. This novelty must be proven, which is difficult with a pure system description, as there are no generally accepted concepts for system comparisons. Therefore, scientific publications often focus on which new tasks can be solved with the system or which new methods have been used. The system and its architecture take a secondary role, which in turn makes the publication of system comparisons more difficult.

These principal problems lead to a very limited exchange of information on the architecture of robot systems via scientific publications. The complete architecture of complex robotic systems is usually only known to the direct developers of these systems.

Even in robotics challenges such as the Amazon Picking Challenge or the Darpa Subterranean Challenge, in which similar robot systems have to solve predefined,

identical tasks and are evaluated on this basis, little can be learned about the architecture of these systems despite extensive publications. Comparative work, e.g. Eppner et al. [40], use individual aspects to categorize the systems, but a systematic analysis with regard to the selected architectural approaches is not possible because this knowledge is not accessible. The following quote illustrates this problem for the Darpa Robotics Challenge (DRC) and the available publications:

> *"The descriptions of the DRC robotics software systems generally provide implementation-specific solutions without reference to a general robotics system architecture."* — Backes et al. [5]

The *RoAF* is based on the systematic description of the architecture of specific systems. For the reasons mentioned above, however, this description is not based on existing publications. Instead a process is defined that enables the robot architect himself to create a systematic architecture description.

2.4 Architecture Frameworks according to ISO 42010

In the software community, especially for distributed systems, the challenges are very similar to those in robotics. The systems here are also complex, exhibit a high degree of heterogeneity and are often subject to continuous development.

In order to enable the comparability and systematic development of these systems, the concept of *Architecture Description* was developed. This allows systems to be described in terms of their *Architecture*. In order to harmonize *Architecture Descriptions* within a domain, the concept of the *Architecture Framework* was introduced. Both approaches have been standardized and are defined in ISO 42010. There is a large number of *Architecture Frameworks* that apply this standard. A current overview can be found on the following website: http://www.iso-architecture.org/ieee-1471/afs/frameworks-table.html

Some of these *Architecture Frameworks*, such as the 4+1 View Model [65] or the Siemens 4 Views [57] refer to software architectures. However, the concept of *Architecture Frameworks* has also been transferred to other domains. The Zachmann Framework [147] for example is applied to enterprises and their structures.

The concept of *Architecture Frameworks* has also been transferred to the field of technical systems. The ESA Architecture Framework [48] is used to describe the *Ar-*

chitecture of ESA space missions, e.g. the Galileo navigation system. The *Architecture Framework* was also applied to IoT concepts with IEEE Std 2413 [58].

For robotics, no *Architecture Framework* has yet been defined. However, the objectives of an *Architecture Framework* largely cover the requirements of robotics. The *RoAF* therefore defines an ISO42010-compliant *Architecture Framework* for the robotics domain, which is described in detail in Chapter 4.

2.5 Chapter Summary

In this chapter, related work on the *RoAF* was presented. This was divided into several directions. In the first section, the classical robot architectures were presented. Subsequently, the different variants of robotic software frameworks were presented. Three systems were then used to illustrate the problems of extracting architectural knowledge from system publications. Finally, work outside the robotics community was presented that uses the ISO42010 *Architecture Framework* tool to capture the *Architecture* of a wide variety of systems.
The *RoAF* transfers this tool to the robotics domain. The following chapter therefore presents the foundations of the *RoAF*, the approach of the *Architecture Framework* and the explicit description of the robotics domain.

Chapter three

Foundations of the Robot Architecture Framework

In order to design a process for the systematic description of *Robot Architectures*, two fundamental questions must be clarified.

The first question is what is meant by the term *Architecture*. As described in the previous chapter, there is some work in robotics that deals with the *Architecture* of *Robots*. However, depending on the category of work, different aspects are considered. Classic architectures deal with the question of which concepts control the behavior of a *Robot* and how these concepts are combined. Software frameworks expand the term to include approaches for organizing software components, and for many system papers, the system architecture is simply a block diagram with important high-level components and their relationships. It is therefore difficult to find an all-encompassing definition for the *Architecture* of *Robots*. Therefore, other fields were searched for appropriate approaches. In the field of software engineering, the ISO 42010 standard "Systems and software engineering – Architecture description" defines both a general architecture concept and the elements necessary for describing an architecture. The questions regarding the architecture term and the methodology for describing an architecture are therefore answered in this thesis with the definition from ISO 42010. Hence, this chapter introduces the concept of the ISO 42010 and its components and terminology.

The second question to be answered is what constitutes a *Robot*. The aim of this work is not to draw a sharp distinction between *Robots* and other complex systems, but to

find a definition that represents the commonality of all *Robot* systems. This serves as a basis for using the *RoAF* to describe a wide range of different *Robot Architectures*. This chapter therefore introduces an inclusive definition for *Robots*. Furthermore, the terminology used and the commonalities of robotic systems are presented.

3.1 Architecture Frameworks

In software engineering, it is generally recognized that the design of complex systems requires architectural concepts. Especially for distributed systems, approaches have been developed to systematically describe the *Architecture* of complex software systems. This enables the comparison of software systems, but also supports the design of new systems and the further development of existing systems. The approaches developed for this purpose have been standardized and are described in ISO standard 42010 [59].

Since the properties of an *Architecture Description* are also suitable for the requirements of robotics and since autonomous *Robots* also partly consist of distributed software systems, an *Architecture Framework* for the domain of robotics is developed in this thesis.

The concept of the *Architecture Framework* and its terminology are therefore introduced below. A central term here is the concept of *Architecture*, which is defined in ISO42010 as follows:

Def. 3.1 Architecture:
fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution. [59]

According to this definition, every system implicitly has an *Architecture*. In order to compare systems and approaches, an explicit description of this *Architecture* is required.

Architecture Description For this reason, the ISO 42010 standard deals with the way system *Architectures* can be expressed, which is referred to as *Architecture Description*.

In Figure 3.1 the elements of an *Architecture Description* according to ISO42010 [59] and their relationships to each other are shown. The first element in the description of an *Architecture* is the identification of the *Stakeholders* of the system. A *Stakeholder*

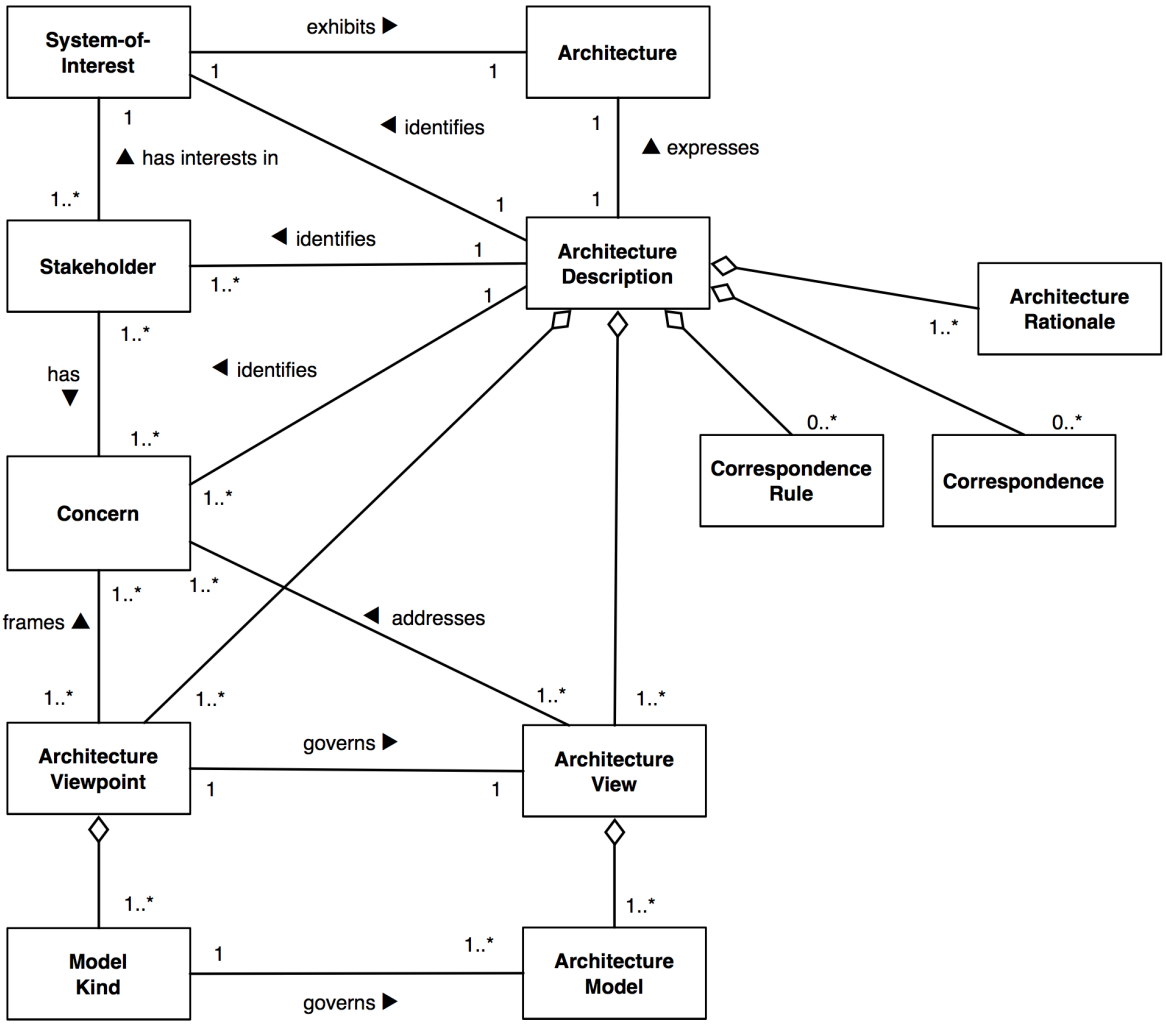

Figure 3.1: *As shown in this figure from ISO42010 [59], an Architecture Description consists of different elements that have defined relationships to each other. The Architecture Description describes the Architecture of a System-of-Interest. It identifies Stakeholders and Concerns. Based on the Architecture Viewpoints Views describing the Architecture are created. (source: ISO42010 [59])*

is any party that has an interest in the system, e.g. developer, user, architect. These *Stakeholders* each have *Concerns* that describe the *Stakeholder*'s interest in the system, e.g. security, maintainability. Each *Concern* in relation to the system must be assigned to at least one *Stakeholder*.

To subdivide the *Architecture Description*, IS042010 introduces the concept of the *Architecture Viewpoint*. This *Architecture Viewpoint* is used to describe systems in relation to a specific set of *Concerns*. To formalize this description process, so-called *Model Kinds* can be defined for *Architecture Viewpoints*. These specify how the system's *Architecture* is to be described.

To obtain an *Architecture Description*, the *Architecture Viewpoint* and the *Model Kinds* it contains are applied to a system. This creates *Architecture Models*, which together result in the *Architecture View*. An *Architecture Description* therefore essentially consists of a collection of *Architecture Views*. In addition, so-called *Correspondences* and *Correspondence Rules* can also be described in an *Architecture Description*. These are used to explicitly describe the relationships between the *Architecture Views*.

Architecture Framework There is a one-to-one relationship between the *Architecture Description* and the system. Therefore, each system has its own *Architecture Description*. To harmonize different *Architecture Descriptions*, ISO42010 introduces the concept of the *Architecture Framework* for systems within a domain. In Figure 3.2 the components of an *Architecture Framework* are depicted according to ISO42010.

Domain-specific *Stakeholders*, *Concerns*, *Architecture Viewpoints* and the associated *Model Kinds* are identified. Correspondence rules can also be defined. An *Architecture Description* adheres to an *Architecture Framework* if all *Architecture* elements of the *Architecture Framework* are contained in the *Architecture Description*.

Relation of Architecture Description and Architecture Framework In Figure 3.3 the different levels of abstraction that connect a robotic system with the *RoAF* are shown.

First of all, every system has an *Architecture* according to Definition 3.1. This consists of the fundamental design decisions and concepts that define the system. In order to make the *Architecture* accessible, it must be described. ISO42010 defines the *Architecture Description* approach for this purpose. As these descriptions can be created individually for each *Architecture* and therefore for each system, there is a risk that the same architectural approaches will be described in different ways. The extent and the aspects considered can also vary at arbitrary levels. This makes it difficult to systematically describe and compare system *Architectures*. ISO42010 therefore defines the concept of the *Architecture Framework*, which makes it possible

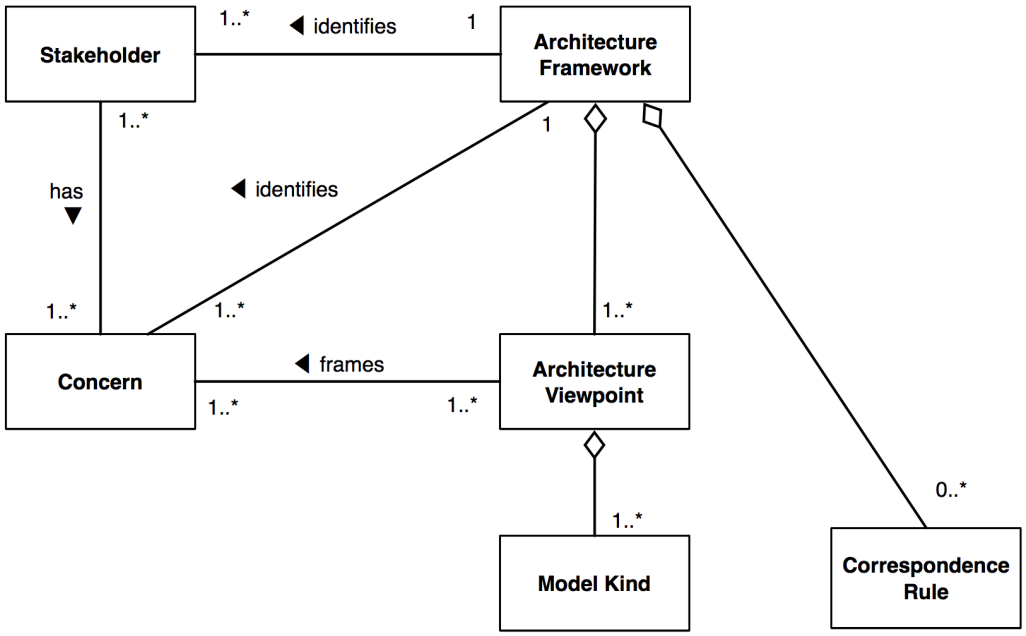

Figure 3.2: *An Architecture Framework according to ISO42010 uses the common definition of elements of the Architecture Description to harmonize the descriptions of a domain. As shown in the graphic from the ISO42010 [59], this involves the identification of domain-specific Stakeholders, Concerns and Viewpoints.*

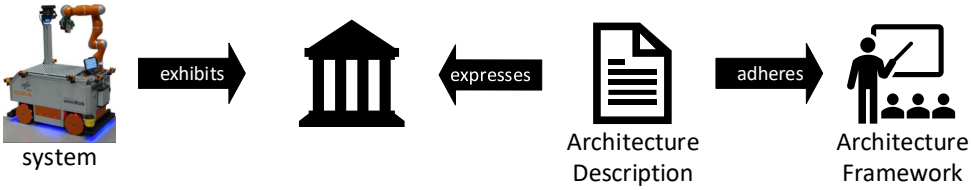

Figure 3.3: *Every system has an Architecture. This can be expressed by an Architecture Description. The Architecture Description itself can adhere to an Architecture Framework in order to achieve comparability with other descriptions of the same domain.*

to harmonize *Architecture Descriptions* for a domain. The *RoAF* is an *Architecture Framework* according to ISO42010 for the domain of robotics. A more detailed consideration of *Architecture Description* and *Architecture Framework* can be found in the referenced standard ISO42010 [59].

3.2 Robotic Domain Definition

The *RoAF* is intended to cover the robotics domain, therefore a definition of robot is needed. ISO 8373 defines a robot as follows: “programmed actuated mechanism with a degree of autonomy to perform locomotion, manipulation or positioning". The robot is defined by using the term autonomy. In ISO 8373, autonomy is defined as “ability to perform intended tasks based on current state and sensing, without human intervention”.

This is a complex definition as it is difficult to distinguish robots from other complex machines. Due to the heterogeneity of robots, it is usually still possible to find counterexamples that do not fit the definition, and vice versa, machines that fit the definition without being perceived as robots.

For the formulation of *RoAF*, a definition of the robot is necessary to define the domain. All robot systems should be included in this definition, but it is not a problem if other machines also fall under this definition. Therefore, the definition is simplified to determine the domain of robotics:

Def. 3.2 Robot:
A robot is a machine which is designed to solve various tasks in physical environments.

Based on this definition, the *Robot* domain is discussed in the following sections.

3.2.1 System boundaries: Fulfilling tasks in the physical world

According to the definition of *Robot*, these systems solve *Tasks* in the physical world. The mandatory relationship between *Task* and the physical world results in a twofold division. The *Task* either consists of observing the state of the world or changing the state of the physical world. A *Task* can therefore be defined as follows for the robotic domain:

Def. 3.3 Task:
A defined modification of the physical world or a determination of information about the physical world.

All *Tasks* can be abstracted either as a modification of the physical world or as a determination of information about the physical world. It should be noted that the *Robot* itself is part of the physical world, so that a desired change to the *Robot* can also define a *Task*. Determining information usually requires an action by the *Robot*, which corresponds to a change in the physical world. Changing the physical world without current information about it is also not possible in a targeted manner. *Tasks* therefore usually require a combination of action and perception to be solved.

Robots are built to solve various *Tasks* in physical environments. The term *Mission* is introduced to describe the entirety of the *Tasks* that a *Robot* should perform during an application.

Def. 3.4 Mission:
The collection of all tasks a robot has to solve.

There are therefore two system boundaries for every *Robot*. There is a boundary between the *Mission* and the *Robot* through which all *Tasks* are assigned. For accomplishing the *Mission* the *Robot* has to solve all these *Tasks*. And there is a boundary between the system and the physical world. From a conceptual point of view, it is very important to separate the *Mission* and the physical world. Although *Tasks* include the physical world in their definition, a *Task* itself is not a physical object and therefore not part of the physical world. Hence, a *Mission* is also not a physical object. Many *Missions* aim to change the state of the physical world, for example, to tidy up a room or to assemble a device. However, this is not the case for all *Missions*. The *Mission* of a soccer *Robot*, for example, is to win the game. To do this, many *Tasks* are necessary to perceive the physical world, where is the ball, where is the opponent or to change the world, e.g. to shoot the ball into the goal. However, the *Mission* as a whole does not aim to change the physical world. Whether the game was won or lost

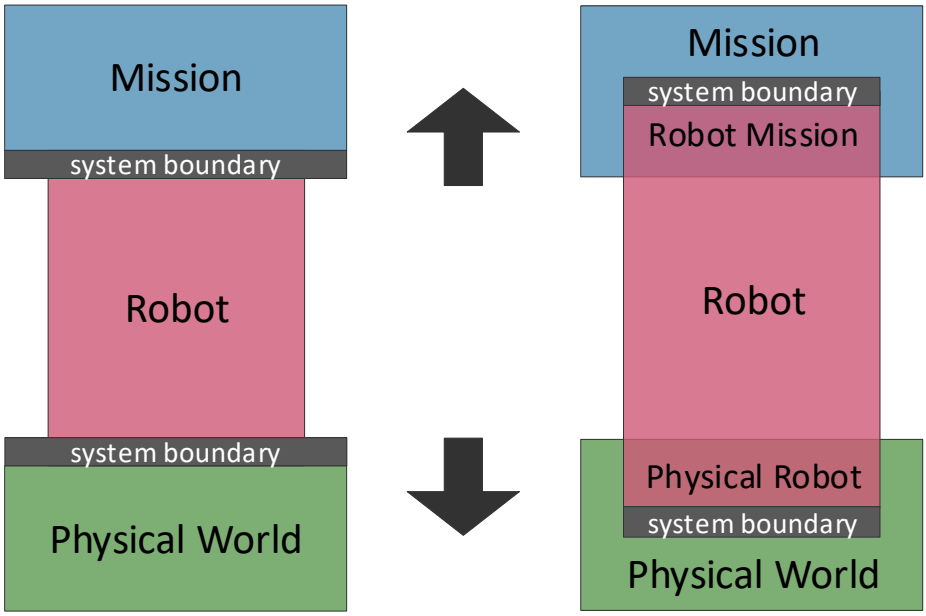

Figure 3.4: *As shown on the right Robots are bridging elements between the Mission and the physical world. Every Robot has a system boundary to both. As indicated by the arrows the Robot itself is also part of the Mission and the physical world, the system boundaries lie within these. Therefore, as depicted on the right, every Robot is part of the Mission, called Robot Mission and part of the physical world, as Physical Robot.*

cannot be determined from the final state of the physical world. Therefore *Robots* can and should also take on such *Missions*.

Every *Robot* can therefore be considered as a bridge element between the *Mission* and the physical world, as shown in the left of Figure 3.4, which is a foundation of the *RoAF*. A closer look at the boundaries leads to an refinement of the diagram, as the blocks overlap. As shown in Figure 3.4 on the right, the *Robot* is part of the physical world and also part of the *Mission*. Therefore, the system boundaries are in the physical world and in the *Mission*.
For the physical world, it is obvious that the *Robot* is also part of the physical world, since a *Robot* consists of hardware components. Each hardware component of the machine belongs to both the *Robot* and the physical world. The disjoint set of the physical world is referred to as *Physical Environment*. For most of the physical world, it is obvious to which set an object belongs, but there are some exceptional cases, such as tools, where a clear assignment is difficult.
For the *Mission*, the boundary is *Mission* and *Robot* specific. Since each *Robot* can be controlled externally to a certain degree, there is generally a part of the *Mission*

that belongs to the *Robot* and an external part called the *Mission Environment*. The *Mission Environment* and the *Robot Mission* can be closely linked, e.g. in cooperative assembly *Tasks*, but there are also systems in which the *Mission Environment* is limited to sending a command to the *Robot*. Many *Robots* receive a *Task* description with a certain level of detail from the *Mission Environment* while the *Robot Mission* is the detailing or planning based on this description.

3.2.2 Domain requirements: flexibility, dependability, and usability as concerns of the user

In the last section, the two system boundaries that all *Robots* exhibit were identified, thus forming an important structure for the domain of robotics. In this section, general requirements for the properties of *Robots* will be identified, which forms a further characteristic of the domain.

Although *Robots* are generally less efficient and more expensive than other machines, they are considered for various applications:

- No other machines are available
- The development of a special machine does not pay off
- A machine should fulfill different tasks
- Other machines cannot solve the application due to the *Task* and/or the environmental characteristics

The common reason for using *Robots* in these cases is their flexibility. Therefore, flexibility is the main requirement for all *Robots*. This flexibility is already included in the definition: A *Robot* is a machine designed to perform **various** *Tasks* in physical environments. Both system boundaries at the *Mission* and the physical world, therefore require flexibility. The degree of flexibility varies between the different categories of *Robots*, but for a *Robot* system as a universal machine, a higher degree of flexibility usually increases the benefit of the system.

The second main requirement for *Robots* is their dependability. In principle, this requirement applies to any machine, but it is much more complex for *Robots* than for other machines due to the flexibility of *Robots*. Standard machines or automatic machines have a *Task* to perform in a clearly defined environment. The machine is always able to perform this *Task* in the same way, which is the basic concept of

these machines. If something does not work, the machine is broken and needs to be repaired. To increase dependability, every step is optimized. With *Robots*, the situation is different because the *Task* and the physical environment are usually not fixed. Depending on the *Task* or state of the physical world, the *Robot* must adapt its approach. It is often not possible to define the best execution procedure in general. During execution, the *Robot* must decide what to do, how to react to events and how to recognize errors. If the same approach to machine dependability is applied to *Robots*, the complexity will be unmanageable as the combination of possible approaches is enormous. Nevertheless, a dependable system is crucial, and therefore strategies to cope with this complexity are a key requirement for *Robots* in general.

The third main requirement for *Robots* is usability. Similar to dependability, this is a general requirement for machines, but the flexibility of *Robots* leads to a complexity that requires special attention to strategies to ensure usability. As shown in Figure 3.5, the complexity of systems increases with increasing flexibility. The most flexible *Robots* are humanoids, but their use requires a lot of effort and a team of robotics experts. This leads to the interesting problem that highly complex systems are theoretically very flexible, but practically difficult to handle even for experts. As a result, most of these systems are only used in a small number of demonstrations that do not even begin to cover the potential of a human scale flexibility. Theoretically flexible systems are therefore not flexible today because they are difficult to operate. On the other hand, special *Robots* such as the Roomba are easy to operate, but their flexibility is very limited. A contradiction between two main requirements: Flexibility and usability is one of the main problems in robotics.

In summary, it can be said that three main requirements for the robotic domain can be identified for the system properties: flexibility, dependability and usability. This applies to every *Robot* and is therefore an important part of the characteristics of the robotic domain.

3.2.3 System components: the parts robotic systems consist of

After the general requirements for *Robots* were discussed in the last section, this section will look at the similarities between the systems themselves. There are very different *Robots*, but groups of components can be identified that are relevant for every *Robot*.

The components of every robotic system can be divided into two groups, the hardware

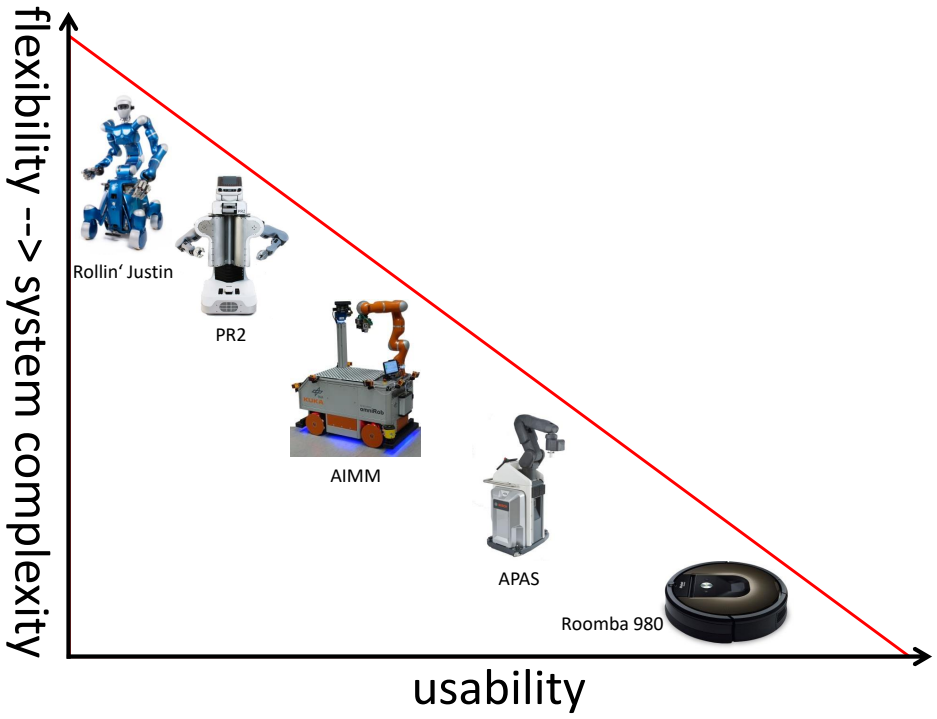

Figure 3.5: *This illustration shows the correlation between Flexibility and Usability. Complex systems, such as humanoids, are necessary to achieve a high level of Flexibility. However, this Complexity leads to low Usability as a lot of expert knowledge is required. (adapted from Dömel et al. [35])*

components and the software components. Even if the components themselves are individual for each system, certain component classes occur in every *Robot*. Thus, these define the domain of robotics. The categories of general *Robot* components are presented in the remainder of this section.

Hardware

Since a *Robot* has at least one *Task* to perform in the physical world, every *Robot* has hardware components. These mechatronic components can be divided into four classes, some of which overlap: Actuators, sensors, data processors and data transmitters.

Actuators The mechatronic components that contain the actuators of a *Robot* form its structure. A *Robot* usually consists of several of these components, such as manipulators, mobile platforms and grippers. The actuators and their

mechanical structure determine the kinematics and dynamics of the system. They therefore represent an upper limit for the *Robot*'s abilities. The use of modular concepts makes it easier to adapt the system to *Tasks* or environments, e.g. by replacing the gripper. However, changes to the actuators have a major impact on the overall system, which is why these components should be designed to cover the entire spectrum of intended *Tasks* and expected environments.

Sensors Sensor components enable a *Robot* to adapt reactively to external and internal influences. This increases the flexibility of the *Robot* and therefore sensors are fundamental components of any robotic system. There are different types of sensors. Some sensors are tightly coupled to the actuators, such as joint sensors, and measure the current state of these devices, which is part of the internal state of the *Robot*. Other sensors, such as cameras, primarily detect the state of the environment, which is an external state. Some sensors are also actively operated, e.g. an optical zoom or a pan/tilt unit. Similar to actuators, sensors define an upper limit of the *Robot*'s abilities and must therefore be selected according to the intended *Tasks* and the expected environment.

Data processors Sensors and actuators are the direct link to the environment. In order to process the information obtained and calculate actions to solve the *Task*, each *Robot* requires computing resources and therefore contains computing components such as computers, FPGAs and GPUs. The amount of computing resources required varies depending on the complexity of the *Task*, the environment and the degree of autonomy, but autonomous *Robots* in particular usually contain multiple computers to provide the required computing power. Some *Robots* utilize external computing resources such as GPU clusters.

Data transmitters In addition to the hardware components required to data processing, a *Robot* must transmit data. A distinction is made between internal communication, which connects the modules of a *Robot* with each other, and external communication, which extends beyond the system boundaries.

Internal data transmission is usually realized via an IT infrastructure such as ethernet or bus systems, which is why components such as ethernet cables, adapters and switches must be integrated. The main requirement for this infrastructure is the ability to transmit the data recorded by the sensors and the subsequent processing steps.

External transmitters can be realized with different devices depending on the modality used. One approach is to use the sensors and actuators as input and

display devices, e.g. for teach-in or for gesture-based commands. Additionally *Robots* usually provide dedicated hardware devices for external data transmission. These are often buttons, displays, status lights and acoustic signals. For more detailed information, an IT interface such as an ethernet connection or a wireless access point is often integrated.

Software

Today's *Robots*, especially those with a high degree of autonomy, are software-intensive systems. Software is the most important tool for achieving the flexibility required by *Robots*. In recent decades, the modularization of robotic software has been one of the main drivers of progress in robotics. Systems with the complexity of the mobile manipulator AIMM [35] typically have more than 100 software processes running in parallel to implement the system behavior. This section provides an overview of the most important general software components that constitute robotic systems.

Interfaces to sensors and actuators These software modules are the interfaces to the hardware components of the *Robot*. The sensor modules provide the sensor data and the functions for configuring the sensor. The actuator modules enable access to the actions of the hardware and the information provided by the hardware.

Perception Sensor data processing takes place in this module class. Low-level sensor data processing modules, such as stereo processing, work directly with the sensor data stream and require only a few parameters for their work. As a rule, they deliver the processed data as a stream. Other important modules of the perception class are modelers and object detectors. These higher-level modules provide various data at a higher level of abstraction, e.g. object positions or geometric representations of the environment.

Knowledge representations Perception of the environment is not enough to be able to act independently. Additional information is required to solve *Tasks*. This is why modules store information about the state of the world, objects and processes, for example, and make it available to the system.

Sequence, task and mission control These modules combine various other software modules to perform a *Task*. By employing hierarchies, the complexity of the system becomes manageable.

Planner The solution of complex problems using the knowledge of the *Robot* is performed by planner components. Typical representatives of this module class are global path planners or view point planners for exploration.

Human-robot interfaces These modules are used for communication with humans. The basic interface is a status display that shows the status of the *Robot*. An interactive interface is required to teach the system, which can be implemented using a GUI. More complex interactions between *Robots* and humans to solve a *Task* cooperatively require more sophisticated interfaces, such as intention recognition.

Middleware Communication between different software modules is usually solved through the use of middleware. Depending on the requirements, there are different solutions for large data streams, real-time communication or more flexible communication.

Development tools In order to build a complex system, tools are required that are not necessary for the use of the *Robot*, but for its development. For example, viewers are used to visualize the internal state of the robot.

3.3 Chapter Summary

The foundations of *RoAF* were presented in this chapter. These are divided into two aspects.
First, the concept of the *Architecture Framework* was presented and the associated terminology introduced. The *Architecture Framework* defines the methodology chosen for the *RoAF* to create a systematic *Architecture Description*. In addition, the concept of the *Architecture Framework* makes it possible to harmonize the *Architecture Descriptions* of a domain. The *RoAF* is therefore an *Architecture Framework* for the domain of robotics.
The second aspect of this chapter was therefore the description of the robotics domain. For this purpose, definitions and terminologies were introduced in order to precisely formulate the scope of the *RoAF*.
Building on this, the next chapter develops the *RoAF* and its components.

Chapter four

Robot Architecture Framework

This chapter defines an *Architecture Framework* according to ISO 42010 for the field of robotics. An *Architecture Framework* is a structure that harmonizes the process of an *Architecture Description* for a domain. The defined *Architecture Description Elements* support the description of the individual *Architecture*, as a defined structure can be used. In addition, an *Architecture Framework* facilitates the comparison of different *Architectures*, as the description styles are similar. An important aspect of an *Architecture Framework* is that it should be applicable to all systems in the domain. A *RoAF* must therefore be suitable for the *Architecture Description* of all *Robots*.

The ISO standard specifies the elements of an *Architecture Framework* with *Stakeholders*, *Concerns*, *Architecture Viewpoints* and *Correspondences*. To describe any *Robot Architecture* these *Architecture Framework Elements* (*AFE*) itself are *Architecture* agnostic.

In this chapter, these elements and their relations are defined for the domain of robotics. Every *Architecture Framework Element* is referenced with an identifier when introduced. First, the robotic *Stakeholders*, i.e. people who have an interest in *Robots*, are identified. Based on this, the central *Concerns* of the robotic domain are determined. Based on these elements, the *Architecture Viewpoints* of the robotic domain are derived. Finally, the connection between the various *Architecture Viewpoints* is shown in the *Correspondence Rules*.

4.1 Stakeholder

Stakeholders (*AFE 1*) are mandatory elements of any *Architecture Framework*. There are two categories of *Stakeholders* in robotics: the group of *Stakeholders* who are interested in the system because of its application, and the group of *Stakeholders* who are interested in the development of the system. In the following, these groups are referred to as *User* and *Developer*.

User The *User* (*AFE 1.1*) is interested in using a *Robot*. The system itself is not modified by the *User*, but the *User* uses the system's interfaces to make the *Robot* do the intended *Tasks*. There are different members in the *User* group. The responsibilities of the roles vary depending on the *Robot* type. However, the different parties are present in almost all *Robots*.

Operator (*AFE 1.1.1*) Person who switches the *Robot* on and off, acknowledges emergency exits and starts programs.

Coworker (*AFE 1.1.2*) The human worker shares the environment with the *Robot*. In more complex scenarios, the human worker also shares *Tasks* with the *Robot*, resulting in collaboration between human and robot.

Maintainer (*AFE 1.1.3*) Personnel responsible for maintaining the *Robot*, including e.g. calibration procedures and recovery from faults.

System Integrator (*AFE 1.1.4*) Persons who configure the *Robot* to perform the intended *Tasks* in the desired environment.

(Factory) Planner (*AFE 1.1.5*) Those who plan the use of the *Robot* (e.g. in an industrial production facility) taking into account the abilities of the system.

Owner (*AFE 1.1.6*) People who purchased the *Robot*.

A human worker can take on several of these roles, e.g. operator, coworker and maintainer. The *User* is usually not interested in the internal structure of the *Robot*, but in its application. Depending on the *Robot Architecture*, however, the *User* must understand the internal structure of a *Robot* in order to be able to program a *Task*, for example. This is not practical for complex *Robot* systems, as a large amount of expert knowledge is required to understand such systems.

Developer The development of such usable systems is the task of the *Stakeholder* group *Developer* (*AFE 1.2*). The *Developer* must be able to deal with the complexity of the system. One approach to this is the separation and distribution of concerns to different experts. The following specialists can be identified for an autonomous *Robot*, who are therefore part of the *Stakeholders* of the *RoAF*.

Mechanical Developer (*AFE 1.2.1*) Person interested in the mechanical design of the *Robot*.

Electronics Developer (*AFE 1.2.2*) Person who develops and integrates the electronic components.

Software Module Developer (*AFE 1.2.3*)) Person who develops software modules for the system.

Software Infrastructure Developer (*AFE 1.2.4*) Software developer who provides the software infrastructure.

Software Integrator (*AFE 1.2.5*) Person who integrates software modules into the system.

Skill Developer (*AFE 1.2.6*) Person who uses the *Capabilities* of the system to implement *Skills*.

Application Developer (*AFE 1.2.7*) Person who develops generic applications for the system.

Robotic System Developer (*AFE 1.2.8*) Person who coordinates the various specialists for a *Robot*.

Robot Architect (*AFE 1.2.9*) Person who creates and analyzes concepts for *Robots*.

As with the *User*, one person can also take on several roles in the *Developer* group. As already mentioned, the common interest of the *Developer* is to develop a *Robot* that meets the needs of the *User*. Therefore, the *User*'s *Concerns* are also the *Developer*'s *Concerns*, as these determine the development process. In addition, the internal structure of the system is of fundamental interest to the *Developer*, which leads to further *Concerns*.

4.2 Concerns

In this section, the *Concerns* (*AFE 2*) of the *RoAF* are identified. The *Concerns* of an *Architecture Framework* are linked to *Stakeholders*, as shown in Figure 3.2. Different *Stakeholders* may have different *Concerns* of a domain. They are also linked to the *Viewpoints*. A *Viewpoint* addresses a set of *Concerns*, whereby a *Concern* can also be addressed by several *Viewpoints*. In the following, the *Concerns* of the *RoAF* are identified and then the relationships to the *Stakeholders* are listed. The relationship to the *Viewpoints* is established later in this chapter in the definition of the individual *Viewpoint*.

4.2.1 Concern Identification

As the standard defines neither the granularity nor the type of *Concerns*, these can be freely selected according to the requirements. The first *Concerns* result directly from the general requirements to a *Robot* and are introduced as *Robotic Concerns* (*AFE 2.1*). As identified in Subsection 3.2.2, there are three main requirements for a *Robot* from the perspective of the *User*. Every *Architecture* must therefore address the following three aspects.

Flexibility (*AFE 2.1.1*) *Robots* solve different *Tasks* by definition. How this flexibility can be implemented is a universal *Concern*.

Usability (*AFE 2.1.2*) *Robots* should solve *Tasks*. Which *Tasks* these are is determined externally. Every *Robot* must therefore have usability, which makes it a universal *Concern*.

Dependability (*AFE 2.1.3*) *Robots* must perform their *Tasks* reliably and safely despite the wide range of possible applications. This also applies to every *Robot*, which makes it a universal *Concern*.

As already mentioned in Chapter 3, a higher flexibility of a system usually leads to a higher complexity. As depicted in Figure 4.1, the complexity of a system can affect the dependability and usability of a system. While this is not an inevitable consequence of complexity, considerable effort is required to ensure the usability and dependability of a complex system. One of the most important strategies in robotics is the use of autonomy to create complex but usable and dependable systems.

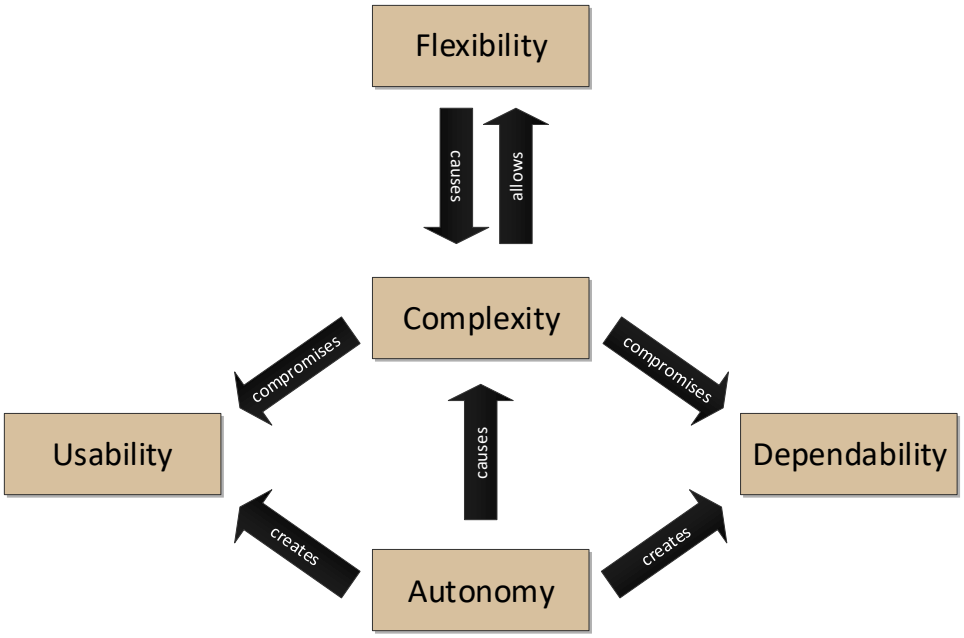

Figure 4.1: *The five Robotic Concerns have relations to each other. As depicted the central Concern of every autonomous Robot is the Complexity. A higher Flexibility of the system often leads to a higher Complexity. For example, the use of additional sensors increases the Flexibility of a system, but also leads to a higher Complexity. This Complexity is in turn the cause of error sources, which impairs Dependability and also reduces Usability, since new aspects such as the lighting conditions must be taken into account when operating the Robot. Autonomy can help to increase Usability and Dependability again. For example, the system can calibrate its sensors autonomously. However, autonomous functions also increase the Complexity of the Robot.*

Therefore, the following *Concerns* are added to the *RoAF*:

Complexity (*AFE 2.1.4*) *Robots* are complex systems that have many internal and external dependencies. Mastering complexity is a challenge for every *Robot*, which makes it a universal *Concern*.

Autonomy (*AFE 2.1.5*) *Robots* must fulfill contradictory requirements. They should be flexible but at the same time usable and dependable. In order to fulfill these conflicting requirements, each complex *Robot* itself must contribute to solving *Tasks*. This makes autonomy a universal *Concern*.

These five *Concerns* - *Flexibility*, *Usability*, *Dependability*, *Complexity* and *Autonomy* - are the key aspects for the *Developers* and thus for the *RoAF*.

Every system in the robotics domain must take these *Concerns* into account, which is why they are referred to as *Robotic Concerns*. However, this does not mean that

all these *Concerns* are relevant for every component of the system. Therefore, each *Stakeholder* can and should only have a subset of these *Concerns*. This separation of concerns, which reduces the complexity for each *Stakeholder*, is a core idea of the *Architecture Framework* according to ISO 42010.

In addition to these *Robotic Concerns*, relevant *Concerns* from the field of software engineering can be identified and are introduced as *Software Concerns* (*AFE 2.2*). For some years now, modularization through component-based design has been used in robotics to control system complexity, increase flexibility and simplify the reuse of components. As a result, today's *Robots* are highly distributed software systems. Therefore, the *Concerns* of this category of software systems can be transferred to robotics. The *4C* introduced by Radestock and Eisenbach [96] are a well-known approach for distributed software systems and have already been applied to robots, e.g. in [26]. They identify relevant aspects from the perspective of the *Robot* as a complex software system:

Coordination (*AFE 2.2.1*) is concerned with the interaction of the various system components. [96]

Configuration (*AFE 2.2.2*) determines which system components should exist, and how they are inter-connected, and is based on principles of software architecture. [96]

Communication (*AFE 2.2.3*) deals with the exchange of data, with a foundation of communication paradigms such as request-reply, synchronous and asynchronous. [96]

Computation (*AFE 2.2.4*) is concerned with the data processing algorithms required by an application, with a foundation in traditional paradigms such as functional programming and object-oriented programming. [96]

Therefore, the *4C*, *Computation*, *Communication*, *Configuration* and *Coordination* are added as *Software Concerns* to the five *Robotic Concerns* of the *RoAF*, see Figure 4.2.

These *Concerns* are central elements of the *RoAF*. As they were derived from the general requirements on *Robots* and from known approaches to software architecture, they can be applied to all *Robots*.

Architecture Concerns of the robotic domain	
Robotic Concerns	Software Concerns
• Flexibility • Usability • Dependability • Complexity • Autonomy	• Coordination • Configuration • Communication • Computation

Figure 4.2: *The Architecture Concerns of the robotic domain are formed by the five Robotic Concerns and the four Software Concerns. These 9 Concerns shape the Architecture of each Robot*

4.2.2 Relations between Stakeholders and Concerns

In accordance with ISO 42010, the *RoAF* identifies both the *Stakeholders* of the domain as well as the *Concerns* of the domain. In addition, there is a relationship between the *Stakeholders* and the *Concerns*. As described in the previous section, not every *Concern* has to be applied to every part of the system. Similarly, although the *Stakeholders* all have an interest in the system, they do not all share the same *Concerns*.

This section therefore describes the relationship between the *Stakeholders* and the *Concerns* of the *RoAF*:

Mechanical and Electronics Developer →
Flexibility, Dependability, Usability, Complexity
The main task of the mechatronics developer is to design a system that has the physical ability to implement the required *Flexibility* in its *Tasks* and the physical world. In addition, *Robots* have interfaces to the physical world that ensure *Usability*. Another important aspect is the safety of the system adressed by the *Dependability*. At the same time, the *Complexity* of the system should be as low as possible.

Software Module Developer → *Computation, Complexity* The software module developer designs software components that add new functionalities to the

system based on data. The main *Concern* is therefore *Computation*. Since these components themselves can be very extensive, *Complexity* is also a relevant *Concern*.

Software Infrastructure Developer → *Communication*, *Complexity* A software infrastructure is created to link the modules together, with the focus on *Communication* between the individual modules. The overall design of an autonomous *Robot* usually consists of more than a hundred modules, which together form a complex structure.

Software Integrator → *Configuration*, *Complexity* The integration of the software modules into the *Robot* is essentially a *Configuration* process in which the *Robot*-specific parameter sets are defined. Due to the complexity of the systems and modules and the existing dependencies, *Complexity* is also a relevant topic here.

Skill Developer → *Configuration*, *Coordination*, *Dependability*, *Autonomy*, *Complexity* *Coordination* and *Configuration* at runtime are the main *Concerns* of the *Skill* developer. In addition, the *Skill* developer strives for reliable system behavior and implements *Autonomy* by combining system functionalities. All system components come together here, which leads to a high level of *Complexity*.

Application Developer → *Usability*, *Flexibility*, *Autonomy*, *Complexity* The task of the application developer is to develop an interface to the environment that offers the required *Flexibility* without compromising the *Usability* of the system. The *Complexity* of the system should remain hidden from the *User*.

Robot Developer → *Flexibility*, *Usability*, *Dependability*, *Complexity* As a system engineer, the *Robot* developer primarily has the requirements of the *Users* in mind, which concern the *Flexibility*, *Usability* and *Dependability* of the *Robot*. It is the task of the *Robot* developer to transfer these general requirements to the various *Stakeholders*. This involves evaluating assumptions to find a suitable compromise between generality and specialization.

Robot Architect → *Autonomy*, *Complexity* The architect develops concepts for realizing the *Autonomy* of *Robots* and for mastering the inherent *Complexity*. For this purpose, general structures and approaches are identified and formulated in *Architecture Descriptions* and *Architecture Frameworks*. These general concepts guide the *Robot* developer in the pursuit of his goals.

This assignment of *Concerns* to different *Stakeholders* shows the potential for simplifying the *Architecture Description*. Certain *Concerns* are not relevant for certain

Stakeholders and therefore do not need to be addressed. This achieves a separation of concerns, which is the basis of every *Architecture Framework*.

The *Concern Complexity* in contrast plays a special role. Each *Stakeholder* of the *Developer* group has this *Concern*. As shown in Figure 4.1, it plays a central role and is linked to all other *Concerns*. This poses a challenge for the formulation of an *Architecture Framework*, because if everything is connected, it is not possible to describe it separately. The *Complexity* of *Robots* is the central challenge and the concepts for dealing with it are a very important element of any *Robot Architecture*. Therefore, the *RoAF* has been extended with the concept of *Model Abstraction Type*, which is introduced in the next section.

4.3 Structure of the Viewpoints

In the previous sections of this chapter, the *Stakeholders* and *Concerns* of the *RoAF* were identified, along with their relationships to one another. Identifying *Stakeholders* and *Concerns* is an important step in any *Architecture Description*, since they have a great influence on the *Architecture* of a system. However, both *AFEs* are outside the architect's sphere of influence. The architect can identify *Concerns* and *Stakeholders*, but cannot decide for or against them. In the following, this, including the *Robot* itself, is referred to as *Architecture Context*.

The actual *Architecture* of the *Robot* is described by using *Viewpoints*. Every *Viewpoint* views the entire system from a defined perspective. This means that the *Viewpoint* are independent of each other. For example, *Viewpoints* can be added or removed without affecting other *Viewpoints*. Four *Viewpoints* have been defined for the *RoAF*, which will be presented in detail in the following sections.

This section discusses the general structure within the *Viewpoints* of the *RoAF*. The ISO 42010 specifies that an *Architecture Viewpoint* uses *Model Kinds* to further subdivide the *Architecture Description*. These *Model Kinds* are then used in the description process to create *Architecture Models*, see Chapter 5.

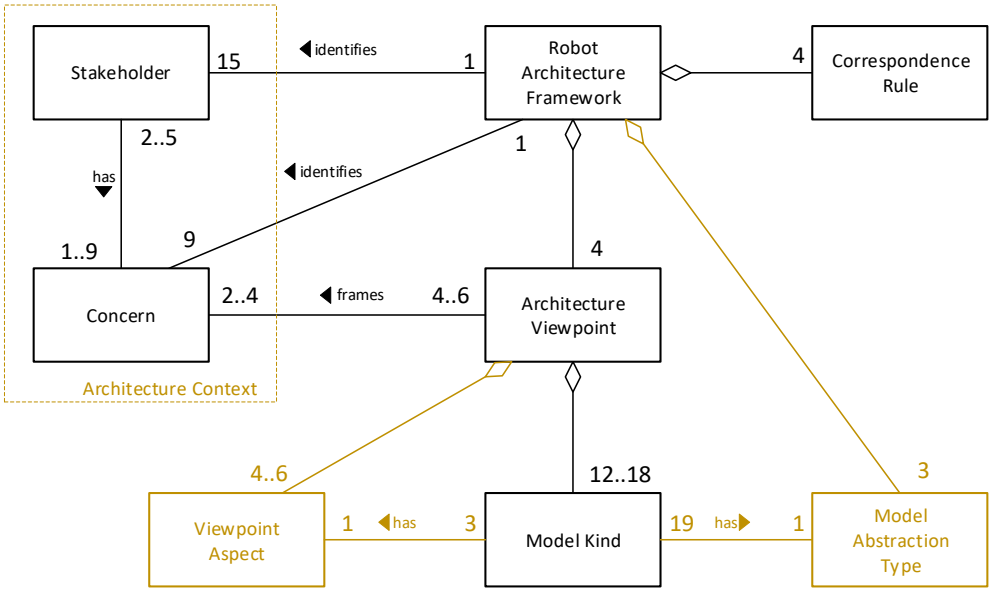

Figure 4.3: *The Architecture Description Elements of the RoAF specify all components required for an Architecture Framework (shown in black). The RoAF identifies a total of 15 Stakeholder and 9 Concerns as well as their relationships, which form the Architecture Context. In addition, the RoAF defines four Viewpoints, each containing 12 to 18 Model Kinds. To capture these in a structured way, two new Architecture Description Element were defined for the RoAF: the Viewpoint specific Viewpoint Aspects and the three independent Model Abstraction Types (shown in brown). (adapted from ISO42010 [59])*

As shown in Figure 4.3, the *RoAF* has been extend with the *Architecture Framework Elements Viewpoint Aspect* and *Model Abstraction Type*. Since all *Architecture Framework Elements* of the *Architecture Framework* are included in the *RoAF*, it is still ISO42010 compliant. The two additional *AFEs* are used to structure the definition of *Model Kinds*. Each *Model Kind* in the *RoAF* has exactly one *Viewpoint Aspect* and one *Model Abstraction Type*. In addition, there must be a *Model Kind* for each combination of *Model Abstraction Type* and *Viewpoint Aspect*.

	Model Abstraction Type A	Model Abstraction Type B	Model Abstraction Type C
Viewpoint Aspect 1	Model Kind 1A	Model Kind 1B	Model Kind 1C
Viewpoint Aspect 2	Model Kind 2A	Model Kind 2B	Model Kind 2C
Viewpoint Aspect 3	Model Kind 3A	Model Kind 3B	Model Kind 3C

Table 4.1: *The Model Kinds in the RoAFs are defined by the Viewpoint Aspects and the Model Abstraction Types. For each possible combination between these two AD Elements, a Model Kind is created in the RoAF. This results in a matrix structure with the dimensions Viewpoint Aspect and Model Abstraction Type.*

As shown in Table 4.1, the *Model Kinds* of the *RoAF* therefore form a matrix. The dimensions are the introduced *AFEs Viewpoint Aspect* and *Model Abstraction Type*. As can be seen from Figure 4.3, the *Viewpoint Aspects* are specific to each *Viewpoint* and are therefore introduced in the respective *Viewpoints* sections. The *Model Abstraction Types*, on the other hand, are independent from the *Viewpoints* and are presented in the following sections.

4.3.1 Model Abstraction Types

To address the *Concern Complexity* the *RoAF* introduces the element of the *Model Abstraction Type* (*AFE 3*). As described in the previous section, the *Concern Complexity* is relevant for every *Stakeholder* and thus for every *Viewpoint*. This makes it difficult to address this *Concern* in the *Viewpoints* without creating a high degree of dependency. For this reason an approach that is orthogonal to the subdivision into *Viewpoints* is developed. The key point is to use abstraction in order to address the *Concern Complexity*. *Model Abstraction Types* define three different abstraction levels in the *RoAF* on which the *Robot*'s *Architecture* is described. In addition, the *RoAF* defines how these abstraction levels are connected to each other. By dividing the *Architecture Description* in multiple levels of abstraction, different types of relationships can be considered and described separately. The three *Model Abstraction Types* of the *RoAF* are presented below.

Model Abstraction Type - Guideline

The highest level of abstraction in the *RoAF* is realized by the *Model Abstraction Type Guideline* (*AFE 3.1*). At this level, the *Robot*'s *Architecture* is seen as a collection of general *Architecture Decisions*. Each individual *Guideline* stands alone and therefore has no dependencies or connections to other *Guidelines*. A *Guideline* refers to at least one *Viewpoint Aspect*. *Guidelines* formulate desired system properties or solution concepts of the *Robot* without taking feasibility into account. These can therefore also contradict each other.

The advantage of this abstraction level is the high degree of subdivisibility of the system. Regardless of the complexity of the system, objectives can be set with regard to specific *Viewpoint Aspects*. This also allows easy transferability to other systems.

In this way, similarities and difference s can be identified at a level of abstraction relevant to the architect.

Example: “The safety of the system is achieved by a safety operator.” At the *Guideline* level, only the goal is defined. How the safety operator achieves safety and what is necessary for this and whether it is possible at all does not play a role for the *Guideline*.

At the level of the *Guidelines*, it is not possible to evaluate whether the *Architecture* works, as contradictions are ignored. Therefore, if the *RoAF* is used to support modification or creation of *Robots* the *Guideline* level is not sufficient.

Model Abstraction Type - Approach

Below the *Guideline* level is the *Approaches* (*AFE 3.2*) level. This level deals with concrete concepts of how the goals of the *Guidelines* are achieved on the system. *Approach* concepts are therefore often more system-specific as they include the basic feasibility. Each *Approach* refers to at least one *Guideline*. Often a *Approach* also refers to several *Guidelines*, which can also contradict each other. The technical implementation of the *Approach* does not play a role at this level of abstraction. The complexity of the *Architecture* is therefore limited to the relationships between the various *Viewpoint Aspects* of the respective *Architecture Viewpoint*. *Approaches* have no connection to each other. However, different *Approaches* can refer to the same *Guidelines*.

Example: “Stopping the motors via a wireless emergency stop” This *Approach* defines how the *Guideline* “Safety is achieved by a safety operator” can be approached. In other words, the *Robot* should stop as soon as a wireless emergency stop is pressed. This takes into account properties of the system, e.g. mobility, as well as the stopping leading to a safe state. This *Approach* is not suitable for a drone, for example.

At the *Approach* level, concepts are formulated on how to apply the *Guidelines*. In addition, different *Guidelines* can be connected here via a common *Approach*. This is particularly interesting for comparing *Architectures* that have identical *Guidelines* but define different *Approaches*. Alternative *Approaches* for identical system requirements are described here. The complexity of the *Architecture Description* is reduced by not linking the *Approaches* between each other. But for the realization of a system, technical constraints and these dependencies between the *Approaches* must be formulated, which is covered by the last *Model Abstraction Type Implementation* presented in next section.

Model Abstraction Type - Implementation

The most concrete abstraction level of the *RoAF* is the *Implementation* (*AFE 3.3*) *Model Abstraction Type*. This is about the technical architectural decisions that enable the realization of the *Approaches*. However, this should not be confused with the description of the implementation, i.e. a system description. The *Implementation Model Abstraction Type* describes the *Architecture*, i.e. the important concepts of a system and not the system itself. A *Implementation* can refer to several *Approaches* and thus formulate the technical dependencies.

Example: " Wireless emergency stop must be connected via secure communication." The *Approach* for using a wireless emergency stop is implemented via this *Implementation*. In contrast to a system description, the specific hardware is not described, but rather an aspect that the hardware must fulfill.

The *Architecture Decisions* of the *Implementation Model Abstraction Type* are of a technical nature. A *Implementation* always refers to at least one *Approach*, but can also refer to several *Approaches*. The concepts and decisions for implementing the hardware and software are documented here. Even at the lowest level of abstraction, the *RoAF* does not describe a system, but the *Architecture* of a system.

Relations between Model Abstraction Types

The different *Model Abstraction Types* and their restrictions create relationships between the *Models* of a *View*.

As shown in Figure 4.4 on the left, the result is that for each *Model Kind* of *Model Abstraction Type Implementation* at least one relationship to a *Model Kind* of *Model Abstraction Type Approach* exists. Similarly, for each *Model Kind* of *Model Abstraction Type Approach* there is at least one relationship to a *Model Kind* of *Model Abstraction Type Guideline*. Since *Model Abstraction Types* does not allow relationships on the same abstraction level, there are no cross-relationships.

In the process of describing an *Architecture* on a *Robot*, see Chapter 5, the *Model Kinds* from the *Architecture Framework* are used to create *Architecture Models* which constitute the *Architecture Description*. These *Models* inherit therefore the abstraction level of their *Model Kind*. The relationships of the *Model Abstraction Types* result in corresponding relationships of the *Models* derived from them. Consequently, a *Model* of a certain *Model Abstraction Type* never has a relationship to a *Model* of the same

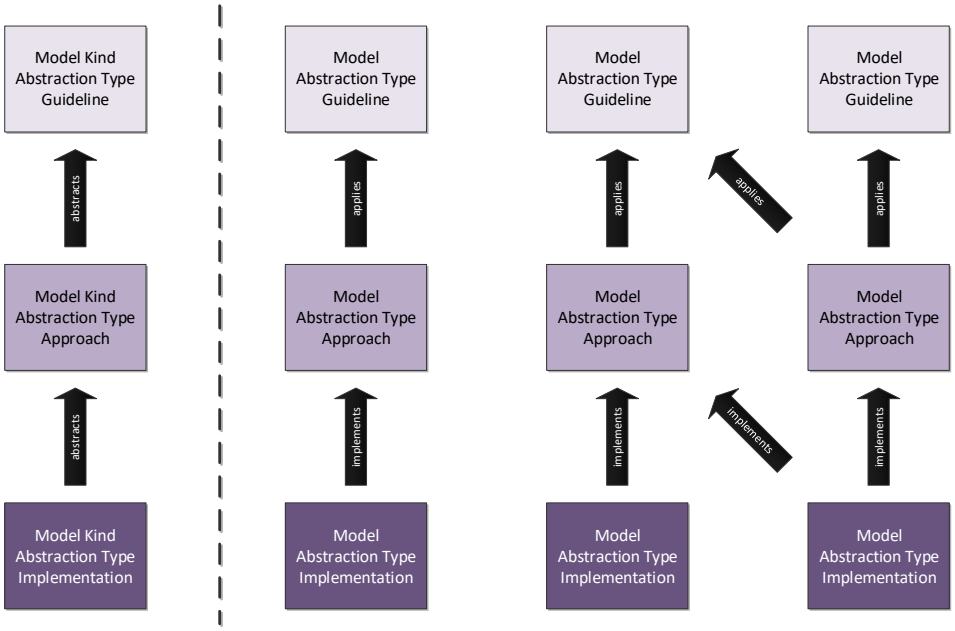

Figure 4.4: *Model Abstraction Types are used to address the Complexity of Robot. Abstraction is used for this purpose. The level of abstraction at which a Model Kind describes the Architecture is determined by the Model Abstraction Type. When they are used to create the Architecture Models, connections are created between these different levels of abstraction.*

Model Abstraction Type. There is also no direct relationship between *Guideline* and *Implementation* level. The simplest case of the relationships is shown in the middle of Figure 4.4. The *Guideline Model* is applied by a *Approach Model*, which in turn is implemented by an *Implementation Model*. In general, however, a *Model* of a higher abstraction level can have any number of relationships to *Models* of the next lower level. Conversely, a *Model* can also have any number of connections to *Models* of the next higher level.

In order to depict the relationships between the various *Models* clearly and make connections visible, each *Viewpoint* of the *RoAF* contains the so-called meta *Model Kind Relations* that represents these relationships graphically. Since the relations between the individual *Models* is mandatory by the *Model Abstraction Types* of each *Model Kind*, the *Models* created with this meta *Model Kind* do not contain any new information regarding the relations, but merely summarizes them in a central location. The *Model Kind Relations* is the only *Model Kind* of the *RoAF* that is used in all *Viewpoints*.

4.3.2 Viewpoint Selection

The *Viewpoints*(*AFE 4*) subdivide the *Architecture Description* into subtopics, which are further detailed by the *Model Kinds*. The selection of the *Viewpoints* is a very important step in the definition of an *Architecture Framework*.

According to its definition, a *Viewpoint* must enable a complete description of the system's *Architecture* with regard to the perspective. If the perspective is chosen very narrow, the *Viewpoint* only describes a very small part of the system. A large number of *Viewpoints* is then required to fully describe the system's *Architecture*. This poses the risk of similar or competing concepts being assigned to different *Viewpoints*. This impairs clarity and makes it difficult to compare systems.

A very coarse subdivision with wide perspectives, on the other hand, jeopardizes the separation of concerns approach. For complex systems, it is then difficult to define the problem context with sufficient precision. The *Architecture Description* then remains at a very general level. However, concretization can be achieved via the assigned *Model Kinds*.

Ultimately, a compromise must be found between comparability and accuracy. Since a *Architecture Framework* is developed for all *Robots* in this work, the number of *Viewpoints* was chosen to be low with four. However, each of these *Viewpoints* defines a large number of *Model Kinds*, which brings the granularity of the *Architecture Description* to a level that is relevant for the *Developer*.

The foundation for the selection of the *Viewpoints* is the general system identification from Section 3.2. As shown in Figure 3.4, each *Robot* connects the *Mission* with the physical world. Two of the *Viewpoints* can be derived directly from this, which are therefore relevant for all *Robots*: The *Robot* is part of the physical world. This aspect of the system is considered under the *Viewpoint Physical*. Since every *Robot* contains hardware according to Definition 3.2, this *Viewpoint* is relevant for all *Robots*. The *Robot* is also part of the *Mission*. The *Robot Mission*, which is realized by the *Robot*, is considered by the *Viewpoint Mission*. This is also part of the *Robot*'s definition and therefore generally applicable to *Robots*. These two *Viewpoints* also cover the two system boundaries of *Robots*.

The remaining two *Viewpoints*, *Capabilities* and *Skills*, are derived from the *4C*. As described in Section 4.2, these can be generally applied to *Robots*. The *Concerns Computation* and *Communication* are addressed in the *Viewpoint Capabilities*. The modules and their data flows can be viewed from this *Viewpoint*. The *Concerns Configuration* and *Coordination* are dealt with in the *Viewpoint Skills*. This perspective focuses on

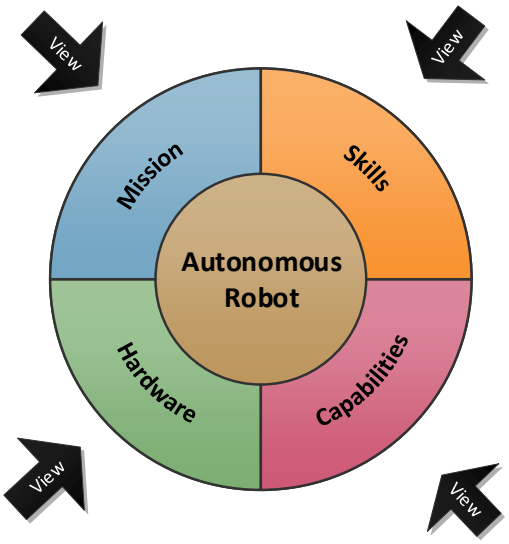

Figure 4.5: *As shown in the graphic, the View of the Architecture depends on the perspective. Each View is a complete description from the respective perspective, but all Viewpoints are necessary to fully grasp the Architecture.*

the parameterization and coordinated execution of software functionalities.

All *Robots* can be examined under these four *Viewpoints*. There are relationships between these *Viewpoints*, which are represented in *RoAF* as *Correspondences*, see Section 4.8. It is important not to confuse *Viewpoints* with a subdivision of the system. Unlike a layer concept, for example, the system is not subdivided into different abstractions with fixed interfaces. The functionality of *Viewpoints* always consists of viewing the entire system from a certain perspective.

This is shown graphically in Figure 4.5. The *Architecture* of an autonomous system cannot be viewed directly due to its complexity. However, the *Viewpoints* provide the perspectives from which the *Architecture* of the system can be viewed from different angles. Only the four *Views* together provide a complete picture of the *Architecture* of the autonomous system.

4.4 Viewpoint Physical

The *Physical Robot* is considered under this *Viewpoint*. This includes all mechanical and electronic parts of the *Robot*. In addition, the physical properties such as workspace, dynamic restrictions and payloads are visible from the *Viewpoint Physical*(*AFE 4.1*). Of course, the system boundary to the physical environment is also visible from this *Viewpoint*.

Framed *Concerns*	Typical *Stakeholder*
Flexibility	mechanical developer
Usability	electronic developer
Dependability	*Robot* developer
Complexity	*Robot* architect

Table 4.2: *Concerns and Stakeholders of the Viewpoint Physical*

As shown in table 4.2, in addition to the *Stakeholders* of the overall system, the *Robot* system developer and the *Robot* architect, the hardware developers are the *Stakeholders* of the *Viewpoint Physical* and correlate with the corresponding *Robotic Concerns*. The *Software Concerns* of the software domain do not apply here.

4.4.1 Viewpoint Aspects

As shown in Figure 4.1, all these *Concerns* are directly linked to each other. In *Viewpoint Physical*, however, the possibility of compensating for complexity through autonomy is missing. More flexibility usually leads to more system complexity, which impairs the usability and reliability of the system. The aim of the *Viewpoint* is therefore to describe the compromise found between the opposing *Concerns*. These *Concerns* are therefore only of limited use for a subdivision into *Model Kinds*. However, in order to achieve an effective subdivision, *Viewpoint Aspects* are defined based on the general system components in Subsection 3.2.3. These six *Viewpoint Aspects* of the *Viewpoint Physical* are presented below.

Structure (*AFE 4.1.1*) The *Viewpoint Aspect* Structure deals with all *Architecture Decisions* that affect the physical structure of the *Robot*. This includes decisions about

the kinematics, the actuators, but also higher-level aspects such as the choice of material or the system weight.

Sensors (*AFE 4.1.2*) The *Viewpoint Aspect* Sensors addresses the *Architecture Decisions* regarding the selection of sensors. In addition to the actual sensor hardware, this also includes the arrangement of these components and the resulting perception areas.

IT (*AFE 4.1.3*) The *Viewpoint Aspect* IT addresses the *Architecture Decisions* regarding the IT components of a *Robot*. This includes decisions on which and how many computers are used. A second important aspect is how these are connected to each other. Other components such as FPGA units or graphics cards also fall under this *Concern*.

Power (*AFE 4.1.4*) The *Viewpoint Aspect* Power addresses the *Architecture Decisions* regarding the power supply of the system. This includes battery selection and concepts for power supply units and power circuits of the *Robot*. Also general *Architecture Decisions*, e.g. low energy consumption, are also described here.

Safety (*AFE 4.1.5*) The *Viewpoint Aspect* Safety addresses the *Architecture Decisions* regarding the safety of the system. These can be dedicated components or concepts such as emergency stop systems. However, it can also be implicit concepts such as low system weight.

Interface (*AFE 4.1.6*) The *Viewpoint Aspect* Interface addresses *Architecture Decisions* regarding the interface to the environment. These can be displays or status indicators such as LEDs. However, concepts such as the use of the manipulator itself as an input device can also be documented.

The *RoAF* therefore defines the *Viewpoint Aspects* (VP-A) for the *Viewpoint Physical*:

VP-A1 Structure

VP-A2 Sensors

VP-A3 IT

VP-A4 Power

VP-A5 Safety

VP-A6 Interface

	Guideline	*Approach*	*Implementation*
VP-A1 Structure	VP-M1G	VP-M1A	VP-M1I
VP-A2 Sensor	VP-M2G	VP-M2A	VP-M2I
VP-A3 IT	VP-M3G	VP-M3A	VP-M3I
VP-A4 Power	VP-M4G	VP-M4A	VP-M4I
VP-A5 Safety	VP-M5G	VP-M5A	VP-M5I
VP-A6 Interface	VP-M6G	VP-M6A	VP-M6I

Table 4.3: *Matrix of Viewpoint Physical Model Kinds*

4.4.2 Model Kinds

The *Model Kinds* then result from the *Viewpoint Aspects* and the *Model Abstraction Types*. As shown in Table 4.3, the *RoAF* contains 18 *Model Kinds* for the *Viewpoint Physical*. Each *Model Kind* has a defined *Model Abstraction Type* and a defined *Viewpoint Aspect* from the *Viewpoint*.

4.4.3 Viewpoint Summary

From the *Viewpoint Physical*, the *Robot* is viewed as a physical system. The *Viewpoint Aspects* Structure, Sensors, IT and Power are derived directly from the hardware classes identified in Chapter 3. The *Viewpoint Aspect* Safety addresses the *Concern Dependability*. The system boundary to the *Physical Environment* is covered by the *Viewpoint Aspect* Interface. The 18 *Model Kinds* of the *Viewpoint Physical* then result from the combination with the *Model Abstraction Types*.

4.5 Viewpoint Capabilities

Autonomous *Robots* are software intense systems, which means that software is a important aspect for *Robots*. The *Viewpoint Capabilities*(*AFE 4.2*) therefore deals with the *Architecture* with regard to the software of the system. Since the software of autonomous *Robots* is complex, a further specialization must be made by choosing a perspective that only considers partial aspects of the software. Two of the *Soft-*

ware Concerns, *Communication* and *Computation*, are chosen for this separation of concerns.

Capabilities are software functionalities that have similarities with the functionalities of hardware components. Every hardware component has at least one functionality that solves a specific problem, such as the battery that supplies the *Robot* with power through chemical reactions. The same applies to software components that provide functions. For example, a stereo processing component is able to generate depth information from two camera images using a correlation-based method.

The *Capabilities* of a system and their relationships to each other define the *Robot* on the software side, similar to how the functionalities of the hardware components characterize the *Physical Robot*. In contrast to hardware modifications, software modifications are very cost-effective. In addition, there are fewer restrictions such as installation space etc., as the memory capacity of today's IT components offers sufficient capacity. A complex *Robot* therefore usually has significantly more software components than hardware components. This potentially large number of *Capabilities* increases the flexibility of the *Robot*.

The following table provides an overview of the *Concerns* and the typical *Stakeholders* of the *Viewpoint Capabilities*:

Framed *Concern*	Typical *Stakeholder*
Flexibility	software module developer
Complexity	software infrastructure developer
Computation	software integrator
Communication	*Robot* developer
	robot architect

Table 4.4: *Concerns and Stakeholders of the Viewpoint Capabilities*

In this section, the term *Capability* is first defined. Then the general structure of *Capabilities* is examined and its types and relationships are presented. Then the *Viewpoint Aspects* and the *Model Kinds* of the *Viewpoint* are introduced.

4.5.1 Definition of Capability

Since, in contrast to the term *Physical*, there is no uniform understanding of the term *Capability*, the term is defined as follows for the *RoAF*:

Def. 4.1 Capability:
A capability is the generation of specific information through computations based on specific data to solve a specific problem.

Several properties can be derived from this definition. A *Capability* is implemented by a software module that has a clearly defined interface in terms of input and output. The goal of a *Capability*, the solution of a specific problem, is clearly defined so that the result of a calculation can be evaluated.
A *Capability* is created, for example, through the use of stereo processing. An algorithm calculates the corresponding depth image based on the incoming image data and parameters. The quality of the depth image can be evaluated using various criteria such as density. Another data-driven *Capability* is, for example, a controller that calculates a control command based on the current joint state to achieve a specific configuration. These sensor data-driven *Capabilities* are often used in sequence to close the action perception loop.
However, there are also *Capabilities* that are not directly related to sensor data. For example, a path planning module calculates a collision-free path based on the *Robot* kinematics, the environment model, the start and goal configuration and various other parameters.

Capabilities create a strong separation of concerns by providing specific solutions. A *Capability* is not intended to solve a problem category completely, but to provide specific information based on specific data using a specific method. For example, the method of correlation-based depth data calculation will not work well if the images are not textured. Nevertheless, the *Capability* will perform a depth calculation with its method based on the provided images. Of course, the *Capability* can provide a quality measure that can be evaluated. The decision on how to react to this situation must be made with further information, see 4.6. The battery does not know why it is supplying energy, nor does the stereo algorithm know for what the depth information is needed.

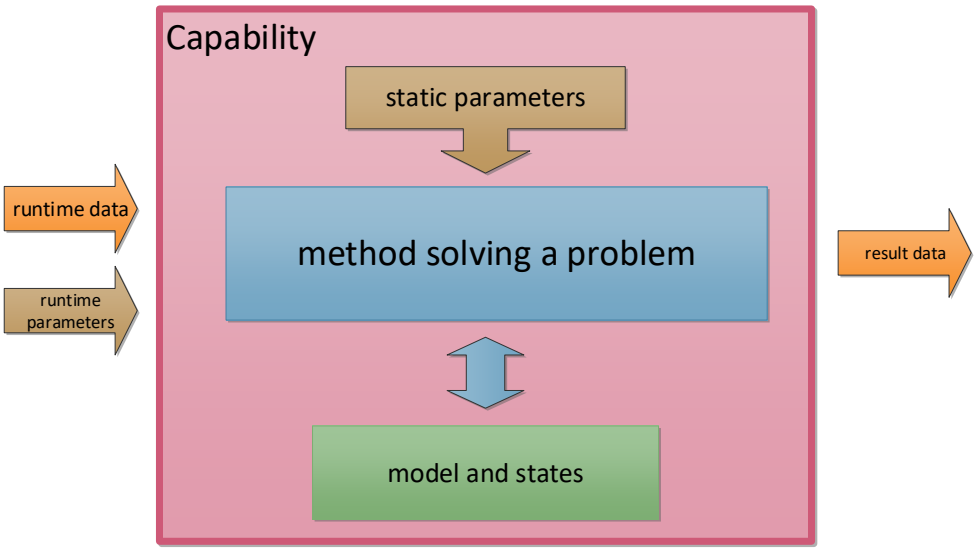

Figure 4.6: *General structure of a Capability: Every Capability is based on a problem-solving method. In the general structure, this forms the central component. The method can be configured by static parameters. Similarly, the Capability can contain models or states that are taken into account during problem solving. The inputs of a Capability consist of the runtime data related to the problem to be solved, as well as optional runtime parameters that can adjust the method at runtime. The output of each Capability consists of the result data.*

4.5.2 Structure of Capabilities

This section describes a general *Capability*. At this high level of abstraction, all *Capabilities* have the same structure, see Figure 4.6.

The central component of a *Capability* is a method that solves the problem, for example an algorithm for stereo data processing. Since *Capabilities* are pure software components, the results of a *Capability* are data containing the solution obtained by the method. In the case of stereo processing, the resulting data is the calculated depth information. In addition, quality measures and timestamps can be provided. A method can potentially have multiple data sources. During execution, the information can be divided into runtime data, i.e. information that is processed by the method, and runtime parameters that configure the method. For the stereo processing algorithm, the runtime data are the images from the stereo camera. Possible runtime parameters are, for example, a region of interest that crops the stereo images. There are also static parameters that are set once when the *Capability* module is started; in this example, a maximum disparity value could be set to speed up processing. In addition, methods often require internal states or models in order to process the data.

For example, stereo processing requires the geometric transformation between the two cameras in order to process depth values. For stereo processing, this model is usually constant. Other *Capabilities* have more complex models. For example, a path planner needs a geometric model of the environment to calculate a collision-free path. If the real environment changes, this internal model must be updated. The models of *Capabilities* often refer to the physical robot and its environment, so there is often an implicit relationship between different *Capabilities*. Internal states can be used to link data at different points in time. For example, a window median filter stores the last values in its internal state to calculate the current median.

Sometimes *Capabilities* are implemented as separate software components, but often, especially for complex modules, different *Capabilities* are provided by the same software module, e.g. sharing a common model. For example, a path planner usually has the *Capability* to plan a collision-free path, but additionally the path planner provides the *Capability* to check a configuration for collisions. On the one hand this increases the performance of the component, on the other hand it reduces the degree of modularity. For example, it is more difficult to replace the collision checker if all components are in one process.

4.5.3 Capability types

Although they have a common structure, different *Capability* types can be identified, which can be distinguished by their purpose and use.

Hardware interfaces

Where exactly the interface between hardware and software is located depends very much on the perspective. An electronics developer might argue that we are still in the software domain as long as the signals are digital. From the perspective of the *Robot* architect, the hardware components are abstracted by including their internal components. The *Hardware Interfaces* thus communicate with the physical components of the *Robot*, possibly in both directions, and make the functions and data of the hardware available to the middleware of the *Robot*.
For example, sensor signals are received by a *Hardware Interface* and movement commands are sent to the actuators. Communication with the hardware components usually takes place via a standardized communication interface such as CAN, Ethernet

or RS-232, which uses a hardware-specific protocol. Therefore, the data inputs and outputs of the *Hardware Interfaces* are only partially visible to the middleware of the *Robot*. *Hardware Interfaces* can provide a standardized middleware interface for different hardware components by encapsulating the hardware-specific functionalities in standardized *Capabilities*. This makes *Hardware Interfaces* crucial for the flexibility of the *Robot* hardware and for the reusability of software components.

Data triggered capabilities

Low-level components are often data-driven, i.e. the *Capability* is always activated when new data is available. An rectification function, for example, rectifies every incoming image. Another common data-driven *Capability* are controllers. A command is calculated on the basis of each new status measurement. Data-triggered *Capabilities* usually have static parameters so that the resulting data is only dependent on the data received. Often, data-triggered *Capabilities* are less computationally intensive, as they should usually deliver the result before the next measurement arrives. The interface of data-triggered *Capabilities* is often based on publish-subscribe mechanisms. The *Capability* is subscribed to data streams that provide the required data. Each time information is received, the method is used to calculate the result and provide the data in a result stream. This allows multiple data-triggered *Capabilities* to build a data processing pipeline.

Event triggered capabilities

High-level components are often event-triggered, i.e. they are explicitly activated by another component, e.g. the flow control. Examples of event-driven *Capabilities* are many path planning *Capabilities*, but also movement commands. This type of *Capabilities* is used for different reasons. Very computationally intensive *Capabilities* are only triggered when the result is really needed. Another reason for this type of *Capability* is if the *Capability* causes a change in the physical world. For example, motion commands are typically provided by event-driven *Capabilities*. A complex and dynamic parameter interface is also a reason for choosing this type. Event-driven *Capabilities* usually use the remote procedure call mechanism. The *Capability* is called by a request. The request contains data, usually the parameters and the data

for the method. Finally, the *Capability* returns its result in a response that contains the result data.

4.5.4 Relations of Capabilities – Concern of communication

Over the last 20 years, the robotics community has turned to component-based design for various reasons. One of the main reasons is the flexibility of the system and the reusability of its components. This simplifies collaboration between institutions and speeds up the development of complex *Robots*, as mature building blocks are available and can be easily integrated. From the architect's point of view, this was a clear advantage of the "separation of concerns" approach, which separates *Computation* and *Communication*. In principle, this development was possible because middleware frameworks such as ROS, DDS, Ice etc. were available to solve the communication problem. However, depending on the *Robot* and its components, different communication requirements have to be met. None of the available middlewares can fulfill all requirements, as the requirements are sometimes contradictory. Therefore, several middlewares are often used in one *Robot*. Some general requirements for communication solutions are listed below.

Real-time Requirements Communication in *Robots* is often subject to time requirements. Depending on the application, these time requirements are more or less strict. Especially for low-level control loops, hard timing requirements must be met to ensure the stability and safety of the system. These hard requirements, which set fixed limits for delay and jitter, are referred to as real-time communication in the field of robotics.

Limitations on bandwidth Autonomous *Robots* are equipped with many sensors that generate large amounts of data. This data must be forwarded to various components, which leads to enormous data traffic in the system. The efficient handling of this data traffic on the hardware used is an important requirement for communication within a *Robot*.

Runtime Flexibility Communication within a *Robot* changes at runtime. A robotics middleware must be able to handle changes in the communication structure, e.g. new components that require data. In addition, components can generate different data depending on their configuration. Therefore, the data streams must also be flexible in terms of data types.

Debugging and Logging Tools Especially for autonomous, reliable systems, the tools of the middleware to examine and analyze the communication between the components are of great importance. The detection of errors in the communication and the logging of the communicated data should be provided by the middleware.

4.5.5 Viewpoint Aspects

The *Viewpoint Capabilities* describes the *Architecture* from the perspective of the software modules and their connection. The relevant *Stakeholders* are therefore the developers of the software modules, the developers of the software infrastructure and the software integrators. The central *Concerns* of these *Stakeholders* are *Computation* and *Communication*. These two *Concerns* are therefore adopted as *Viewpoint Aspects*.

However, since *Robots* are not purely software systems, two *Concerns* relevant to robotics are added. Firstly, the *Concern* of world modeling is a central aspect of robotics. Every *Robot* works in the physical world. Therefore, software components solve problems based on the physical world. In order to solve these with software, the relevant aspects must be modeled. How the *Architecture* of the *Robot* solves these problems is dealt with in the *Viewpoint Aspect* World Model.
The second robotic aspect is the sense-act loop. Every *Robot* perceives its environment via sensors and derives the necessary actions from this, which are executed via actuators. This changes the physical world, which in turn can be perceived via the sensors. This closed loop can be observed in every *Robot* but the approaches to implementation are very diverse. The *Concern* Sense-Act Loop serves to document the *Architecture Decisions* in this regard.

Therefore the four *Viewpoint Aspects* of the *Viewpoint Capabilities* are presented below.

Sense-Act Loop Closure (*AFE 4.2.1*) The *Viewpoint Aspect* Sense-Act Loop addresses all *Architecture Decisions* regarding the closure of the Sense-Act Loop. This can be implemented at very low levels of abstraction, e.g. control. However, complex loop closures, e.g. via a knowledge representation, also fall within the scope of the loop closure *Aspect*.

World Model (*AFE 4.2.2*) The *Viewpoint Aspect* World Model addresses the *Architecture Decisions* regarding the modeling of the physical world. It documents how data relevant to the software modules can be recorded, stored and retrieved. In

	Guideline	*Approach*	*Implementation*
VC-A1 Sense-Act Loop Closure	VC-M1G	VC-M1A	VC-M1I
VC-A2 World Model	VC-M2G	VC-M2A	VC-M2I
VC-A3 Communication	VC-M3G	VC-M3A	VC-M3I
VC-A4 Computation	VC-M4G	VC-M4A	VC-M4I

Table 4.5: *Matrix of Viewpoint Capabilities Model Kinds*

the simplest case, this can be parameters and measured values, but also complex knowledge representations such as ontologies can be used.

Communication (*AFE 4.2.3*) The *Viewpoint Aspect* Communication addresses the *Architecture Decisions* regarding communication between software modules. In addition to the often distributed computer architecture, special challenges in robotics arise in other aspects such as heterogeneity of data, flexibility of data streams and real-time requirements.

Computation (*AFE 4.2.4*) The *Viewpoint Aspect* Computation addresses the *Architecture Decisions* regarding the software methods used in the system. In addition to the basic selection of methods, this *Concern* also addresses the technical interaction of the individual methods or software modules.

The *RoAF* therefore defines the *Viewpoint Aspects* (VC-A) for the *Viewpoint Capabilities*:

VC-A1 Sense-Act Loop Closure

VC-A2 World Model

VC-A3 Communication

VC-A4 Computation

4.5.6 Model Kinds

The *Model Kinds* then result from the VC-A and the *Model Abstraction Types*. As shown in Table 4.5, the *RoAF* contains 12 *Model Kinds* for the *Viewpoint Capabilities*. Each *Model Kind* has a defined *Model Abstraction Type* and a defined *Aspect* from the *Viewpoint Aspects*.

4.5.7 Viewpoint Summary

From the *Viewpoint Capabilities*, the *Robot* is viewed as a system of communicating software components. First, the *Capability* term was defined and introduced. A general structure of this central element in the *Viewpoint Capabilities* was then identified. Various classes of *Capabilities* were then introduced. The linking of *Capabilities* via communication solutions was also described.
The *Viewpoint Aspects* of the *Viewpoint Capabilities* were then identified as *Communication* and *Computation* from the *4C*. These *Software Concerns* were then extended by the domain-specific *Viewpoint Aspects* Sense-Act Loop Closure and World Model. The 12 *Model Kinds* of the *Viewpoint Capabilities* then result from the combination with the *Model Abstraction Types*.

4.6 Viewpoint Skills

This section introduces the *Viewpoint Skills* (*AFE 4.3*). To solve a *Task*, a transition from the declarative to the procedural level is necessary (Volpe et al. [139]). The "what" to do must be translated into a "how" to do it. From the perspective of the declarative level, this is the problem of symbol grounding [52]. Classical logic planners such as Strips [42] are based on the idea that there are actions that deterministically transform the state of the world. The idea is obvious that the *Capabilities* of the system can be used as implementations of these actions, but in a practice a *Capability* will not cause a deterministic change in the state of the world. The effect of executing a *Capability* depends on the context, which is also part of the world state. For this reason, *Capabilities* cannot be used for symbol grounding. However, even with less strict requirements that allow dependencies on the context, the successful execution of a *Capability* does not guarantee that the *Task* has been solved.
An example: The *Task* is to examine an object. To accomplish this *Task*, a movement is performed with a pan/tilt unit equipped with sensors. The movement may be successful, but the object is still not visible because it could be occluded. Continuing the *Mission* would lead to problems because the subsequent steps require the object to be visible. In the worst case, many subsequent steps are executed without the underlying problem being recognized, which makes error detection more difficult. A better approach would be to use a different *Capability* to check whether the *Task* has been successfully resolved. So is it a question of granularity? Can the problem be solved by replacing the *Task* with two more fine-grained *Tasks*, the making visible

itself and then the validation of the visibility?
In practice, this approach is better because the error is detected earlier and error propagation can be avoided. From a conceptual point of view, the procedural layer was forced into the declarative layer, which leads to various difficulties. The biggest problem is that the two *Tasks* together do not result in the *Task* but specify the procedure how the *Task* could be solved. If this procedure does not work, it is difficult to use other ways to solve the problem because the information about the actual goal is no longer available in the description.
Another problem, especially for planners, is that only the sequence of *Tasks* define the state of the world. After the attempt has been executed to make the object visible, the state of the world becomes unknown. The state of the world can only be resolved by executing the next task. The individual steps can therefore no longer be interpreted as the actions of a classical planner. There is therefore a gap between the declarative (*Task*) and procedural (*Capability*) levels. Bridging this gap is the goal of *Skills*. A *Skill* attempts to solve a *Task* by utilizing the *Capabilities* of the system. For example, the *Skill* "Look at object" could call the *Capability* "Move pan tilt", check the visibility with the *Capability* "Viewpoint analytic" and return the result of the *Task* to the declarative layer on this basis. This could be a sufficient *Skill* for a simple system. Complex *Robots* usually offer the possibility of solving a *Task* with many different approaches. For example, the *Skill* of looking at an object could modify the position of the *Robot* to solve the *Task*.

The following table provides an overview of the *Concerns* and the typical *Stakeholders* of the *Viewpoint Skills*:

Framed *Concerns*	Typical *Stakeholders*
Dependability	*Skill* developer
Autonomy	*Robot* developer
Flexibility	*Robot* architect
Complexity	
Configuration	
Coordination	

Table 4.6: *Concerns and Stakeholders of the Viewpoint Skills*

In this section, the term *Skill* is first defined. The general structure of *Skills* is then examined and their types presented. Subsequently, the *Viewpoint Aspects* and *Model Kinds* of the *Viewpoint* are presented.

4.6.1 Definition of Skills

The term *Skill* is widely used in robotics, but there is no generally accepted definition. For *RoAF* we define a *Skill* as follows:

> **Def. 4.2 Skill**:
> A skill is the ability to solve a specific task effectively by a combination of knowledge, capabilities and experience.

This is an abstract definition of the term *Skill*, which refers to the use of the term *Skill* in early work on artificial intelligence, e.g. Sussman [125]. The *RoAF* defines the *Viewpoint Skills* as a perspective in which these task-solving abilities and their implementation are visible. Since each *Robot* has to solve *Tasks*, this *Viewpoint* is generally relevant. The term effective implies that a *Skill* can handle errors through recovery strategies. Many *Robots* have a pick-*Skill*. The *Task* consists of picking up an object. Depending on the *Architecture* and the environment, this *Skill* can be simple, e.g. a pick-and-place unit in a production line picks up the same type of object from a well-defined location. The pick-*Skill* is therefore a hard-coded sequence of movement commands. However, error detection and correction can also be integrated into such systems by triggering a new gripping process if the suction gripper was unable to hold the part on the first attempt.
In more complex systems and in less constrained environments, a *Skill* for grasping objects can be enormously complex. Here, many *Capabilities* such as path planners, object detectors, reachability analysis, often need to be orchestrated in a dynamic environment using domain-specific knowledge and accumulated experience.

4.6.2 Generalized structure of Skills

Often *Skills* are described from a high-level perspective and it is explained how a *Skill Type* can be used, but not how to create a *Skill* or how it is structured. However, the internal structure is at least as important as the interface when creating *Skills*. The structure of a *Skill* depends on its *Skill Type* (see Subsection 4.6.3) and its function. Nevertheless, all *Skill Types* have general elements and structures that constitute them. In Figure 4.7, such an abstracted, generic *Skill* is shown with its components, relationships and interfaces.

From the perspective of the *Mission* (blue boxes), a *Skill* is used to solve a *Task*. The *Skill* can be activated. As long as the *Skill* is active, the *Task* is being solved. Once

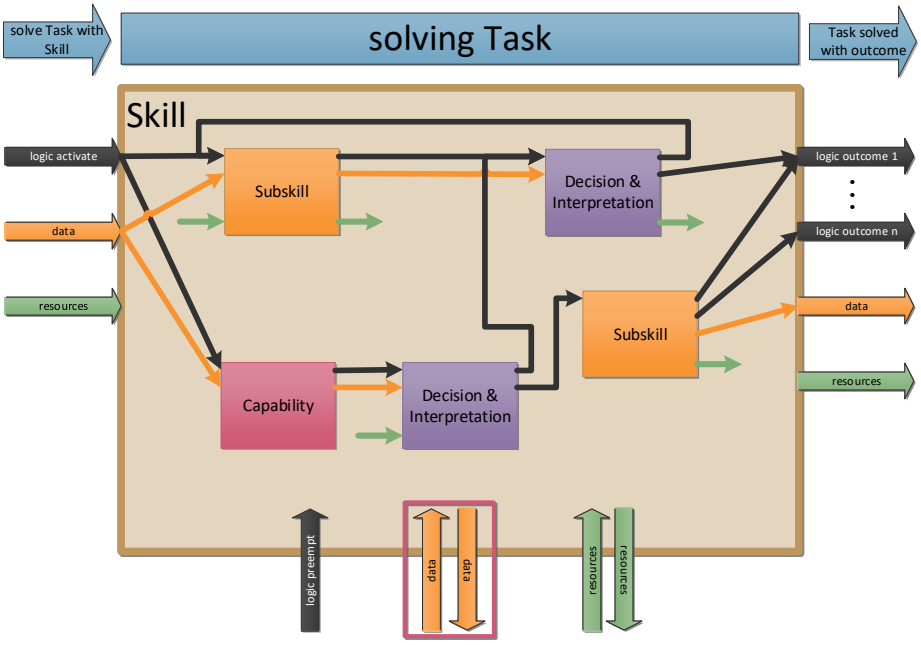

Figure 4.7: *The abstracted Skill outlines the generalized structure of the Skill. A Skill solves Tasks. For this purpose, it uses various components, the Subskills, D&Is components and Capability calls. The connection between these components and the interfaces of a Skill are realised via the three levels logic flow, data and Resources.*

this process is complete, the *Skill* returns, optionally with a declarative result.
The *Viewpoint Skills* shows the strategy of how a *Skill* solves *Tasks*. Generally, a *Skill* consists of several components that are used to solve the *Task*. These components can be divided into three different classes: *Subskills*, *Decisions & Interpretations* and *Capabilities*. In addition, the components are related to each other on three different levels: the logic level, the data level and the resource level.

Relations between Skill components

The components of a *Skill* are connected to each other by various relationships:

Logic flow All components within a *Skill* are connected to each other by logical links (black links). The logical links encode which components are activated depending on the logical results of the previously activated components, e.g. a successfully solved *Subskill*. The logical links contain no further information than the link between results and activations. The logical flow therefore addresses the *Concern Coordination*. The logical flow can create loops by repeatedly activating components. Parallel activation of multiple components is also possible. The connections are therefore $n : n$ connections. A logical result can activate several components and a component can be activated by several results. The activation of a *Skill* leads to the activation of one or more internal components according to the logical links. A *Skill* must implement a logical flow structure that ensures that the overall *Task* is solved by orchestrating its subcomponents.

Data flow In addition to the logical flow, a *Skill* contains data flows between the components and the interfaces of the *Skill*. This data can vary depending on the component and *Skill*. The most common data in *Skill* data flows is configuration data. Therefore, the data flow implements the parameterization of the *Skill* components, which corresponds to *Concern Configuration*. The data flow also contains results of the components, which can be used to interpret information and derive logical results. In addition, the data flow can be used to implement a data processing chain by passing data from one subcomponent to the next. Data flows are $n : n$ connections. A component can receive data from multiple components and deliver data to multiple components. These relationships are often similar to logical relationships, but are generally independent of each other. A *Skill* must implement a data flow structure that ensures that all

subcomponents are correctly parameterized and that all subcomponents are supplied with the required data.

Resource management The third level of connection within a *Skill* is referred to as *Resources*. In contrast to pure software projects, robotics components are often interdependent to a certain degree, as the *Physical Robot* and the *Physical Environment* influences the result of the components. Therefore, a component that is independent from a pure software perspective is indirectly dependent on other components, as their effects change the state of the world. These implicit relationships between components make the development of dependable autonomous *Robots* extremely complex. Therefore, these relationships are formalized via abstract *Resources* to deal with this problem explicitly. For the development of *Skills* it is important to be aware of this additional level, as most components within a *Skill* strongly depend on this concept. The relationships at the *Resource* level differ from the other two relationship types as there is no flow between the *Skill* and its components, but the *Skills* and components interact directly with the *Resources*. A *Skill* must manage the required and produced *Resources* for the *Skill* and its components. Especially with high-level *Skills*, the avoidance of *Resources* conflicts with *Subskills* is an important issue.

Interfaces of a Skill

A *Skill* has four different interface layers (logic, data, *Resources*, *Capabilities*) in three successive phases:

Activation A *Skill* has an input in the logic layer that activates the execution of the *Skill*. In the data layer, any data, often parameters, can be transferred to the *Skill*. In the resource layer, the *Resources* required for the execution of the *Skill* are transferred.

Execution During execution, the *Skill* has a logic input that can be used to stop the *Skill* during execution. There is no interface at the data level during execution, but new data can be accessed and returned with the help of *Capabilities*. These *Capabilities* form the fourth layer of the interface. The *Capability* interface is only used during the execution phase of a *Skill*. *Resources* can be created, used, blocked or consumed during execution.

Result A *Skill* can have several logical outcomes. One of these outcomes is active in the result phase. Any data structures can be returned in the data layer. In the

result phase, *Resources* can be created or released.

Components of a skill

Each *Skill* consists of several components that interact via relationships to solve the *Task*. Three types of components can be distinguished within a *Skill*. Each component type can occur multiple times. Depending on various factors such as *Task*, available *Capabilities*, world knowledge, etc., the design of a *Skill* can vary significantly. Nevertheless, all *Skills* are based on the component types: *Subskill*, *Capability* and *Decision & Interpretation*, which are presented below:

Subskill In particular, high-level *Skills* consist to a large extent of *Subskills* to solve their *Task*. *Subskills* are *Skills* that are integrated as a component into another *Skill*. This component thus enables modularization and the resulting hierarchy helps to create complex *Skills*. With *Subskills*, a *Task* can be broken down into several *Subtasks*, as each *Subskill* also solves its own *Task*.
Another use case of *Subskills* is the creation of *Resources*. The *Task* of the *Subskill* may not be necessary for the higher-level *Task* of the calling *Skill*, but the *Subskill* creates *Resources* that are needed for other components within the *Skill*. The *Subskill* can also be used to generate required data. Furthermore, a *Subskill* can also be used to decide on the further logical flow of the *Skill* based on the logical outcome of the *Subskill*. It is not necessarily the successful solution of the *Subskill* that is of interest, but the information as to whether the *Subskills* could be solved.
For example, a *Skill* for detecting the position of a person in the workspace could return the logical result “failed” because no person was detected. However, this result is desirable for the calling *Skill*, as further steps are dangerous for people in the workspace.

Capability *Skills* have the task of building a bridge between the declarative, *Task*-related level and the procedural level. This goal is achieved by using *Capabilities* as components within a *Skill*. A *Capability* has a different interface than the other *Skill* components, as there is only a very limited logical interface and no resource interface. The logical interface consists of an input and an output. The input activates the *Capability* and the output signals that the *Capability* has returned. As a *Capability* has no *Task*, no additional logical results can be delivered. The data interface is similar to the *Skill* interface. Normally, parameters are given as input and results are returned as output. Since a

Capability does not have a *Resource* interface, the calling *Skill* must ensure that the required conditions are met. The *Capability* component is the only connection to the *Physical Robot*. For example, all perceived data is only accessible via *Capability* components.

Decision & Interpretation The last category of *Skill* components are the *Decision & Interpretation* (*D&I*). These components can generate logical decisions, data and *Resources* based on their input data. The interfaces of the *D&I* components are similar to the interface of a *Skill*. A *D&I* component has a logic input for activation, data inputs and a connection to the *Resources*. The output of a *D&I* component can have several logic outcomes, data outputs and *Resources*. In contrast to *Capabilities* and *Subskills*, *D&I* components are not active over a longer period of time, but decide or interpret their output based on the input. Therefore, no complex calculations take place in these components. The decision can be rule-based, mathematical or based on learned correlations. A typical application of a *D&I* component is the interpretation of the results of a *Capability* component.

4.6.3 Skill Types

Several types of *Skills* can be distinguished, each representing specializations of the general *Skill*. All *Skill Types* therefore have the structure described in the previous sections, but their purpose differs. In this section, various *Skill Types* are presented as examples.

Skill Primitives

The design goal of a *Skill Primitive* is to link a *Capability* with a *Task* to be solved. As shown in Figure 4.8, the *Task* is usually very simple, such as the *Skill* “move-to-touch” introduced by Hasegawa, Suehiro, and Takase [54]. This *Skill* is solving the *Task* "get-into-contact-with-object" by a *Capability* commanding cartesian moments with force sensitivity. Depending on the *Task* and the *Capability*, the result of the *Skill* can be determined by interpreting the result of a *Capability*, e.g. the final position, or based on additional information or tests, e.g. whether an external force was measured.

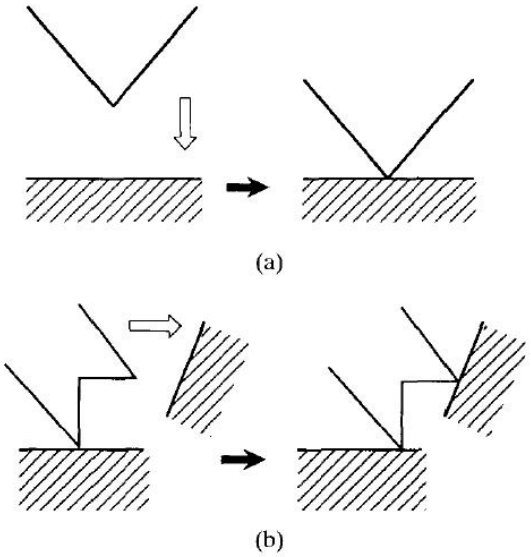

Figure 4.8: *Example of a Skill Primitive: "Move-to-touch" Skill from Hasegawa, Suehiro, and Takase [54]. To solve the Task to get into contact with an object a cartesian movement Capability is used. If a contact force in movement direction is detected the Task is solved. (source: Hasegawa, Suehiro, and Takase [54], © 1992, IEEE)*

The interface of *Skill Primitives* is closely linked to the parameters of the *Capability* used. A *Skill Primitive* cannot provide complex troubleshooting strategies. Preconditions are usually not checked in a *Skill Primitive*, but the context is expected to provide the preconditions. Nevertheless, *Skill Primitives* are very important for dependability as they detect and interpret errors at a fine granularity to avoid error propagation. The *Skill Primitive* is the most basic *Skill Type* and is usually used as a building block for higher level *Skill Types*. In the literature, this *Skill Type* is often not introduced as *Skill*, but as a motion primitive, for example. According to the *Skill* Definition 4.2 of *RoAF*, however, the *Skill Primitive* is a *Skill Type* that bridges the gap between the declarative and procedural levels.

Process Skills

Solving *Tasks* often requires mastering similar processes. For example, many assembly *Tasks* are "peg-in-hole" problems. The design goal of a *Process Skill* is to encapsulate the strategy for solving such a category of problems so that the *Skill* is applicable to any "peg-in-hole" problem. The interface of a *Process Skill* is reduced to the process-related parameters that allow the *Skill* to be adapted to the different problem variants. The other parameters, such as *Robot*-related ones required within the *Skill*,

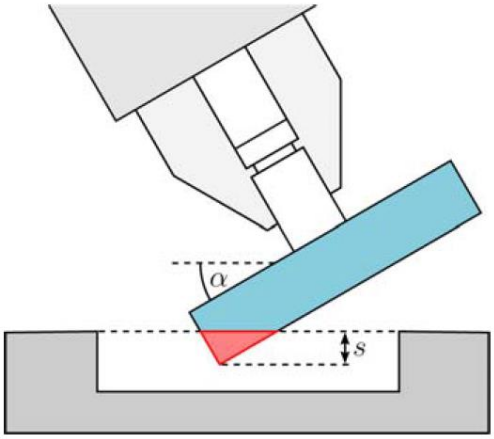

Figure 4.9: *Example of a Process Skill: "Peg-in-hole" Skill with process parameters penetration depth and tilting angle from Stemmer and Bøgh [122]. The Task, which can be solved with this Skill are "Assemble-object-into-hole". Based on these parameters the Skill can be adapted to process variations. The Skill itself then parameterize a set of motion Capabilities used to execute the process and solve the Task. (source: Stemmer and Bøgh [122])*

are either expert knowledge that is integrated during the development of the *Skill* or must be determined using the *Capabilities* of the *Robot*. In Figure 4.9 a *Process Skill* "peg-in-hole" for assembly developed as part of the EU project TAPAS is shown. The parameterization of this assembly process is reduced to two process-related parameters, the penetration depth and the tilt angle, which are necessary to adapt the *Skill* to the *Task* to be solved. All other parameters of the underlying *Skill Primitives* are configured internally.

Process Skills have preconditions, but analogous to the *Skill Primitives*, these preconditions are expected to be fulfilled and are not explicitly checked. One reason for this is that the success of a *Process Skill* strongly depends on the parameterization of the *Skill*. Therefore, the preconditions are of less importance from a high-level perspective, as *Process Skills* does not show deterministic behavior between preconditions and postconditions as long as the parameterization is not appropriate. Therefore, a *Process Skill* cannot be used directly in a symbolic planner. During integration, an adaptation step must be performed to specialize the *Process Skill* to a problem instance.

The parameters of a *Process Skill* are process-dependent, so that finding a parameterization for a defined process solves the same process in different situations. The *Process Skill* "peg-in-hole" becomes a specialized *Skill* "place object of type A in hole of type B". The *Skill Type* thus transfers the expert knowledge to the system itself: Generic approaches to solving *Task* categories using the *Robot*'s *Capabilities* with a focus on the process-relevant parameters. The *Process Skills* thus helps to cope

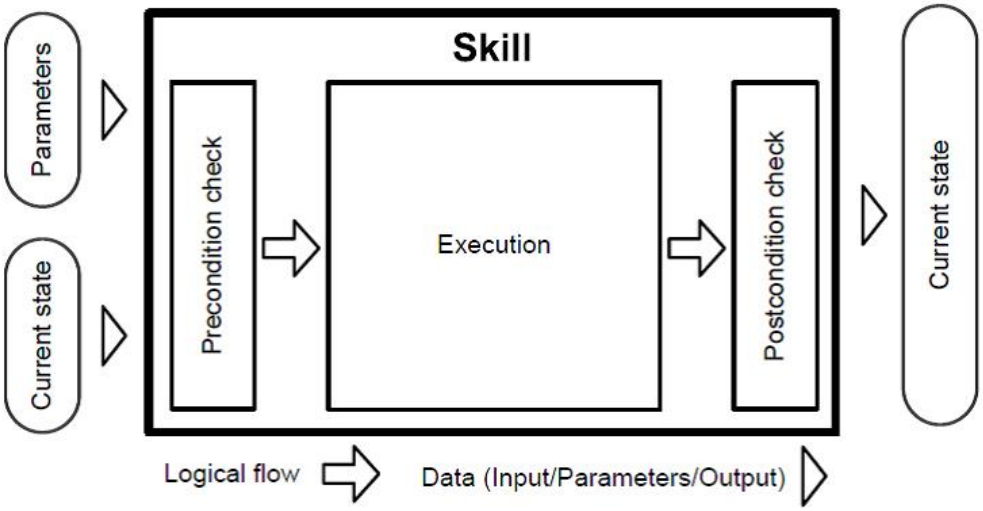

Figure 4.10: *Example of a Task Programming Skill from Steinmetz and Weitschat [121]. The purpose of this Skill Type is to allow intuitive programming of Robots. To achieve this, a Task Programming Skill has only a few parameters that need to be defined. Most of the parameters required for internal Capability parameterization are automatically deduced from the current state. Additional pre- and postcondition checks help the operator to avoid errors. (source: Steinmetz and Weitschat [121], © 2016, IEEE)*

with complexity and increases the dependability of the system by reusing generic, tried-and-tested *Skills*.

Task Programming Skills

A widespread paradigm of future programming concepts for *Robot* applications is programming at *Task* level. This *Skill Type* can be used to implement such an interface. The *Task Programming Skills* represent the abilities of the system and can be easily parameterized by the user to fulfill specific *Tasks*. The aim is to create a user-friendly interface. To achieve this, the parameter set is reduced, interactive methods for parameter specification are used and pre- and postconditional checks are employed to support the user in the development of a functional robot application.

As shown in Figure 4.10, a *Task Programming Skill* is divided into three subcomponents that support the programmer. In the precondition check, the *Skill* checks whether all preconditions are met before execution and helps the user to identify problems. For example, a screwer *Skill* can check whether the screwing tool is mounted. In the second block, the execution, the state-changing operation is carried

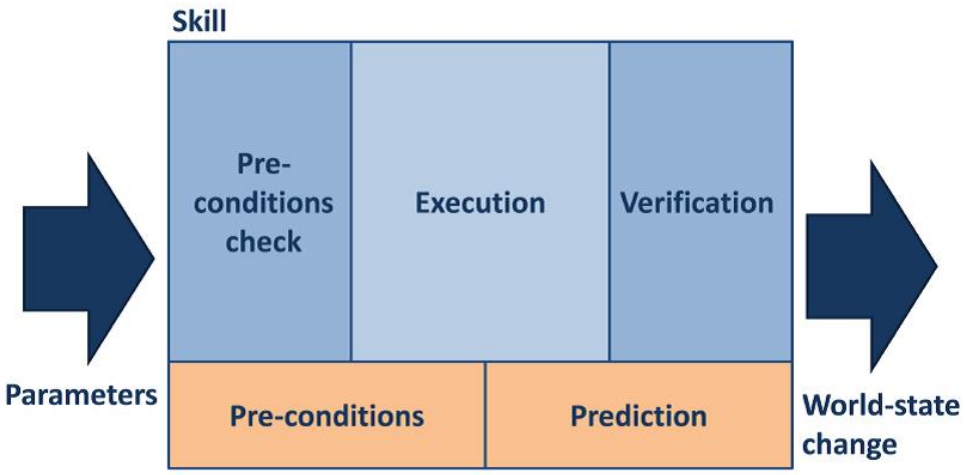

Figure 4.11: *Example of a Task Abstraction Skill from Rovida and Kruger [98] (© 2015, IEEE). This Skill is used to abstract the programming problem to a symbolic level. The execution of the Skill therefore results in a predictable world state change. The goal is to be independent of the current state to allow flexible sequencing of the Skills, e.g. by a symbolic scheduler. Remaining state dependencies are modeled by preconditions. (source: Rovida and Kruger [98], © 2015, IEEE)*

out. During task programming, the human user can help the *Robot* to make decisions and handle errors. The third block is the post-check, which ensures that the *Skill* has completed its *Task* and informs the user if something has gone wrong. *Skills* for *Task* programming represent a direct interface to the user. The user therefore transfers the *Tasks* to the *Robot* by linking the configured *Skills*.

Task Abstraction Skills

Another strategy to create *Robot* applications without programming the *Robot* is to use planners such as SkiRos [97]. These planners are based on a simplified representation of the world, often a state vector, and a set of symbols that can change the state of the world. A major challenge in applying symbolic planners to *Robots* is the problem of symbol grounding, i.e. how this state change is implemented. *Skills* can be used to generate these symbols. The design goal of *Task Abstraction Skills* is therefore to ensure a defined change of world state. The interface of a *Task Abstraction Skill* must correspond to the abstraction level of the planner working at the *Task* level. Therefore, all preconditions of this *Skill Type* must be *Task*-dependent. In addition, the state change of a *Task Abstraction Skill* must be deterministic.

The structure of a *Task Abstraction Skill* is shown in Figure 4.11. The interface are *Task*-related parameters that configure the *Skill* to effect a change in the world state. The formal structure is similar to the *Task Programming Skill* with three subcomponents during execution. The precondition check, the execution step and the verification step. In addition, two blocks are required for the planning interface. A precondition block that the planner tries to fulfill and a prediction block that can be used to predict the resulting change in the world state when the *Skill* is applied. *Task Abstraction Skills* usually close the gap between the declarative and procedural levels at a high level of abstraction, otherwise the world state of the planner becomes too complex and the *Skills* becomes less deterministic. In contrast to the *Task Programming Skills*, it is not necessary to reduce the number of preconditions or parameters for these *Skill Type* as long as they are at the planner's abstraction level. On the one hand, this makes the planner-generated *Robot* application more efficient for complex sequencing problems; on the other hand, all problems below the abstraction level of the planner must be solved by the *Skill* itself.

4.6.4 Viewpoint Aspects

The *Viewpoint Skills* describes the *Architecture* in regard to the aspect of how the available *Capabilities* can be used to solve *Tasks* effectively. In the software community, this aspect is represented by *Concerns Coordination* and *Configuration*. In robotics, these two *Concerns* are strongly intertwined due to the strong coupling of the components through the physical world. For the *Viewpoint Skills*, these two *Software Concerns* are therefore addressed by four *Viewpoint Aspects*, each of which represents a sub-aspect. The identified, generic structure of the *Skill* is used to identify these *Concerns*. In addition to the *Software Concerns*, reliability and error handling are of central importance in robotics. Since *Skills* by definition imply robustness against uncertainties and errors, *Dependability* is a *Viewpoint Aspect*.

The five *Viewpoint Aspects* of the *Viewpoint Skills* are presented below.

Hierarchy (*AFE 4.3.1*) *Skills* already form hierarchies at a structural level via the *Subskill* component. The *Viewpoint Aspect* Hierarchy therefore addresses how coordination and configuration can be achieved through hierarchical approaches. Decisions are documented here as to which hierarchy levels are used in the *Skills* of the *Architecture* and what tasks they have.

Type (*AFE 4.3.2*) The *Viewpoint Aspect* Type addresses the *Architecture Decisions* regarding the selection of the *Skill Types*. All *Skill Types* have different properties and functions. This *Concern* documents why which *Skill Types* is used in a *Architecture* and what the relationship between them is.

Composition (*AFE 4.3.3*) The *Viewpoint Aspect* Composition addresses *Architecture Decisions* regarding the composition of *Skill* components. This includes which *Skill* components are used and how these are linked within a *Skills*.

Resources (*AFE 4.3.4*) The *Viewpoint Aspect Resources* is used to document *Architecture Decisions* regarding relationships between *Skills* and its components at the *Resource* level. The dependencies induced by the physical conditions are modeled here.

Dependability (*AFE 4.3.5*) The *Viewpoint Aspect Dependability* addresses the *Architecture Decision* regarding the reliability of *Skills*. In *Viewpoint Skills*, this refers to the fact that activation of the *Skill* leads to a successful solution of the corresponding *Task*. It also includes concepts of how problems can be recognized and handled.

The *RoAF* therefore defines the *Viewpoint Aspects* (VS-A) for the *Viewpoint Skills*:

VS-A1 Hierarchy

VS-A2 Type

VS-A3 Composition

VS-A4 Resources

VS-A5 Dependability

4.6.5 Model Kinds

The *Model Kinds* then result from the VS-A and the *Model Abstraction Types*. As shown in Table 4.7, the *RoAF* contains 15 *Model Kinds* for the *Viewpoint Skills* . Each *Model Kind* has a defined *Model Abstraction Type* and a defined *Aspect* from the *Viewpoint Aspects*.

	Guideline	*Approach*	*Implementation*
VS-A1 Hierarchy	VS-M1G	VS-M1A	VS-M1I
VS-A2 Type	VS-M2G	VS-M2A	VS-M2I
VS-A3 Composition	VS-M3G	VS-M3A	VS-M3I
VS-A4 Resources	VS-M4G	VS-M4A	VS-M4I
VS-A5 Dependability	VS-M5G	VS-M5A	VS-M5I

Table 4.7: *Matrix of Viewpoint Skills Model Kinds*

4.6.6 Viewpoint Summary

In the *Viewpoint Skills*, the *Robot* is regarded as a system that configures and coordinates software components to solve *Tasks*. Therefore, the term *Skill* was first defined and introduced. A general *Skill* and its components were then identified. As there are different classes of *Skills*, *Skill Types* were introduced. The *Viewpoint Aspects* Hierarchy, Composition and *Resources* were then derived from the general structure of a *Skill*. The various *Skill Types* are addressed via the *Viewpoint Aspect* Type. That *Skill* serve to solve *Tasks* effectively, even when problems occur, is represented in the *Viewpoint Aspect Dependability*. The 15 *Model Kinds* of the *Viewpoint Skills* then result from the combination with the *Model Abstraction Types*.

4.7 Viewpoint Mission

In this section the *Viewpoint Mission* (*AFE 4.4*) is introduced. A *Robot* has to solve, according to its Definition 3.2, *Tasks*. Often this includes several *Tasks* that are dependent on each other. This collection of *Tasks* is referred to as *Mission* according to Definition 3.4.

This collection of *Tasks* can be seen from the *Viewpoint Mission*. The relationship between the various *Tasks* is also visible. In order to fulfill complex *Missions*, the *Tasks* must often be divided into smaller *Tasks*. Another aspect that is visible from the *Viewpoint Mission* is the distinction from the *Mission Environment*.

With finely granular *Mission* descriptions, the *Robot Mission* can be very simple, e.g. the execution of a sequence of *Tasks* in a fixed order. The *Robot Mission* can become very complex, especially with underspecified *Missions* and collaborative *Missions*. A *Mission* such as “tidying the room” in the field of service robotics requires a lot of additional information that must be derived from knowledge databases in order to identify specific *Tasks* that can be performed by a *Robot*. For collaborative *Tasks*, the *Task* is shared between the *Robot Mission* and the *Mission Environment*. Therefore, the *Robot* must know what the complete *Task* looks like and which part of the *Task* is to be executed by the system.

Robotic *Tasks* must be completed in the physical world. They are therefore independent of the *Robot* itself. The dependency exists in the opposite direction; A *Robot* can only fulfill certain *Tasks*, but the *Tasks* can also be completed without the *Robot*. The *Software Concerns* are therefore not relevant for the *Viewpoint Mission*, only the *Robotic Concerns* are.

The following table provides an overview of the addressed *Concerns* and the *Stakeholders* of the *Viewpoint Mission*:

Framed *Concerns*	Typical *Stakeholders*
Dependability	Application developer
Usability	*Robot* developer
Flexibility	*Robot* architect
Complexity	
Autonomy	

Table 4.8: *Concerns and Stakeholders of the Viewpoint Mission*

This section first describes the general structure of *Robot Missions*. As these are closely related to the *Tasks*, they are subsequently discussed. Finally, the system boundary between *Robot Mission* and *Mission Environment* is addressed and the *Viewpoint Aspects* and *Model Kinds* identified are introduced.

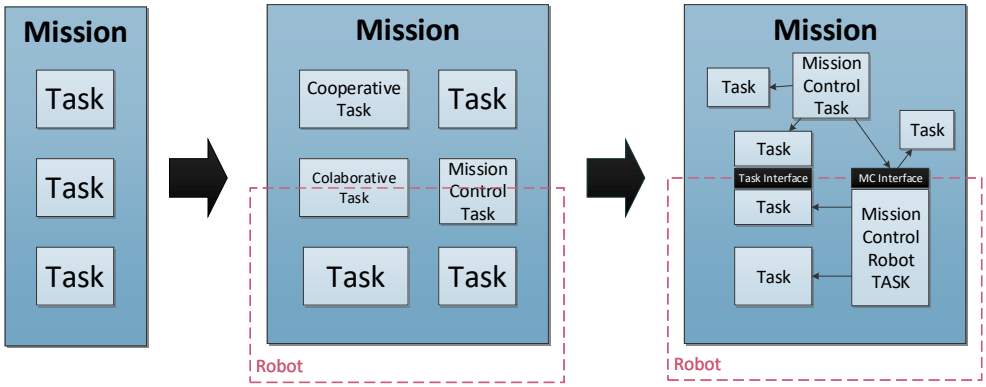

Figure 4.12: *Generalized structure of Robot Missions with increasing level of detail. On the left the Mission is depicted according to the 3.4 as a collection of Tasks. In the middle more details are presented. The Robot covers only a part of Mission. Furthermore there are different type of Tasks like cooperative Tasks located in the Mission Environment or collaborative Task located on the boundary of the Robot Mission. Also ever Robot has a so called Mission Control Tasks which is coordinating the Mission. As depicted on the right two interfaces between Robot Mission and Mission Environment are identified. Furthermore dependencies of the Tasks are illustrated.*

4.7.1 General structure of the Mission

The general structure of *Missions* is shown in Figure 4.12. In its most abstract form, as shown in the figure on the left, a *Mission* consists of a collection of *Tasks*. Since, according to Definition 3.2, each *Robot* must solve *Tasks*, every system can be described from this perspective.

Without losing generality, components can be identified. For each *Robot*, the execution of the *Tasks* must be controlled, i.e. the system must determine when which *Task* is executed. This component is called *Mission Control Task*, as shown in Figure 4.12 in the middle. The *Mission Control Task* is not limited to the system. Part of the *Mission Control Task* is always located outside the system. As a rule, it is determined to a certain extent from outside which *Task* is to be completed and when. A *Robot* can always be activated or deactivated from outside. In the illustration, the system boundary therefore lies within the *Mission Control Task*. *Tasks* can also lie outside the system boundaries. For example, in a *Mission* with several cooperating systems, *Tasks* can be distributed flexibly. *Tasks* that could also be fulfilled by the system lie outside the system boundary if the *Tasks* are distributed differently. For *Tasks* that can only be solved via direct collaboration, the system boundary lies within a *Task*.

This results in two types of visible system boundaries for the *Viewpoint Mission*, as shown in Figure 4.12 on the right.

The first is located within the *Mission Control Task*. It divides this component into the *Mission Control Robot* component, which is inside the system boundary, and the *Mission Control Environment* component, which is outside the system boundary. These components are connected to each other via the *Mission Control Interface*. As each *Robot* is controlled from the outside, each system has a *Mission Control Interface*.

Another aspect shown in Figure 4.12 on the right is the relationship between *Tasks* within a *Mission*. For example, there are *Tasks* that must be completed in a certain order, or that are triggered by an event such as an operator command. Often the *Mission Control Task* manages these dependencies.

For collaborative systems, further system boundaries in *Mission View* may exist. These are defined by the collaborative *Tasks*. For each collaborative *Task*, there is one part of the *Task* that lies within the system boundary and one part that lies outside. The *Task Interface* lies between these two parts and defines the interface between the systems.

The *Mission Environment* can be structured in very different ways. For example, the *Mission Control Environment* can be realized by a human operator or by a higher-level planning instance and can itself be highly complex. For the *Viewpoint Mission*, however, only the components within the system boundaries are relevant. These are: *Tasks*, *Task Interfaces*, *Mission Control Robot* and the *Mission Control Interface* which are presented in the following sections.

4.7.2 Tasks

According to the Definition 3.3, a *Task* is a specific modification of the physical world or a gain of information about the physical world. The *Task* itself is therefore initially independent of the *Robot* but can also be performed by a human or a device, for example. The *Task* itself also has no dependencies but stands alone.

General Structure of Tasks

As shown in Figure 4.13, the execution of a *Task* is triggered. This can be triggered via the *Mission Control Interface* or internally via dependencies between *Tasks*. In this case, there must be a higher-level *Task* that contains the dependency.

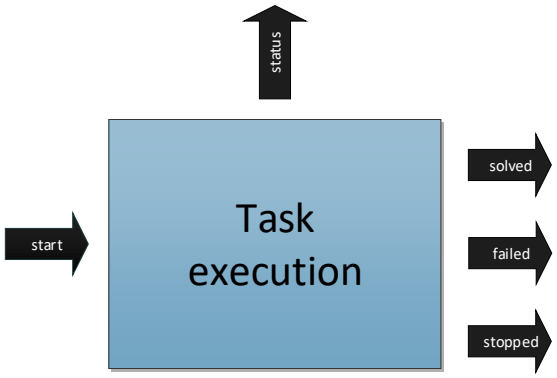

Figure 4.13: *A Task is according to its definition a specified modification of the physical world or a specific information to be acquired from the physical world. Therefore a Task has no parameters and also no data result. The interfaces exist only for the execution of a Task. During execution status information can be published. When exiting the Task execution, the Task is either solved, failed or stopped.*

Status messages can be retrieved as long as the execution is running. An *Task* execution is completed when the *Task* is solved, it has failed or it has been canceled.

A *Task* is defined as a specific modification of the physical world. Therefore, a *Task* also has no parameters with which a further specification is made. Nevertheless, there are *Tasks* that are more similar and *Tasks* that are very different. Similar *Tasks* can often be solved by the same *Skill*, which may require a corresponding parameterization. Conversely, if a parameter is not specified, it is not a *Task*. For example, “drive to a location” only becomes a *Task* if the location is specified.

Task Hierarchies

Tasks have a wide range of complexity. For example, “Move joint 1 to 0 degrees” is a *Task*, as it brings the physical world, in this case the *Robot*, into a certain state. However, a *Task* can also be “Assemble this car”, which is also a modification of the world but is much more complex to solve. Some *Tasks* can be broken down into *Subtasks*. Additional dependencies usually arise between these *Subtask*.

Subtask can have temporal, chronological or causal dependencies. *Subtasks* with a temporal dependency must be executed at a certain time, e.g. every day at 8:00 a.m. *Subtasks* with a causal dependency must be completed when certain events occur, e.g. when someone presses a button. This causal dependency can also refer to internal events, e.g. Pick up the ball when you detect a ball. The third dependency on is the

chronological dependency between *Tasks*. This means that a *Task* should be executed as soon as another *Task* has been successfully completed.

The solution of all *Subtask* always leads to the solution of the parent *Task*, taking into account the dependencies. The sum of all *Subtasks* including the dependencies thus corresponds to the parent *Task*. This distinguishes the *Subtask* from the *Subskill*. With *Skills*, it is not guaranteed that successful execution of all *Subskills* means that the *Skill* is successful.

$$T = \bigcup T_{sub_A} = \bigcup T_{sub_B}$$

$$S \supset \bigcup S_{sub}$$

There are usually various ways to split a *Task* into *Subtask*. Each *Subtask* set corresponds to the parent *Task*. This *Task* decomposition can also be applied to the *Subtasks*, creating a *Task* hierarchy. However, it should be noted that there is a dependency between the *Subtasks* to solve the higher-level *Task*. Accordingly, the *Subtask* decomposition must also take the other *Subtasks* into account. A real division into sub-problems can therefore generally not be solved using the hierarchical approach.

Task States

A *Task* can be in certain states. As shown in Figure 4.14, the initial state of a *Task* is the *todo* state. Further, there are two final states: *solved*, which represents that *Task* has been solved and *failed*, which represents that *Task* cannot be solved. From the initial state exists a direct connection to the state *failed*. This transition can be triggered for example by external events resulting in a not solvable *Task*. Otherwise, an action must be performed to solve the *Task*. In this case, the state of the *Task* is *active*. The *Task* can be solved from this state, which results in the state *solved*, but it can also become unsolvable, which results in the state *failed*. Furthermore it is possible to return from the state *active* to the state *todo*. This happens for example when a action is preempted before solving the *Task*.

Task Interfaces

If a *Task* is not exclusively inside or outside the system boundary of the *Robot*, a *Task Interface* is created. Even if the *Task* is not bound to the system boundary, it is always possible to decide which parts of the *Task* are executed on the *Robot* and which are

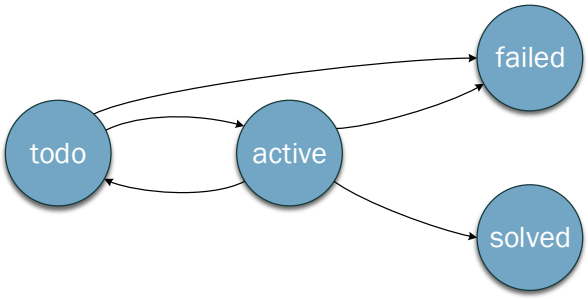

Figure 4.14: *The states of a Task is closely related to the interfaces Figure 4.13. The initial state of a Task is "todo". After the execution has been triggered, it is in the state "active". There are three possible transitions from this state. The terminal state "solved" is reached when the Task is solved. The terminal state "failed" is set when the Task becomes unsolvable. The state "todo" can be activated when the execution of the Task has been stopped. There is also a direct transition between the "todo" and "failed" states when a Task becomes unsolvable without any action.*

not. It is therefore always possible to split a *Task* that crosses the system boundary into *Subtasks*, each of which is completely inside or outside the system boundary of the *Robot*. The two *Subtasks*, which by definition solve a superordinate *Task*, can have different relationships to each other. The different types of *Task Interfaces* are presented below and illustrated using the example of a collaborative mapping *Task*.

1. **Disjunct *Tasks*** The simplest relationship between the two *Subtasks* (Robot/Environment) arises with *Tasks* which can be divided into disjunctive *Subtasks*. I.e. the *Task* is solved together but there is no connection between the two *Subtasks* apart from the joint execution.
   Example: The collaborative mapping of a rover and a drone can consist of the rover exploring the relevant area. At the same time, the drone provides aerial images of the area.
   From the perspective of the *Mission*, there is no connection between the two *Subtasks*. If both are successfully completed, the higher-level *Task* is solved.

2. **Implicit collaboration** In implicit collaboration, the *Robots* solve a *Task* together. However, the collaboration, i.e. the division of the *Tasks*, is not mapped on the *Task* level but arises implicitly via the *Task* itself. A classic case of implicit collaboration is to assign the same *Task* to the *Robot* and its *Mission Environment*. The collaboration arises implicitly via the partial solutions that each system contributes.
   Example: Two rovers are to explore an area. Each rover is given the *Task* to explore the entire area. By combining the respective maps into an overall map and continuously planning the exploration on this map, an implicit collabora-

tion is created that leads to the solution of the overall problem. In this case, too, there are no dependencies between the two *Subtasks* from the perspective of the *Mission*. Both *Task* and the two *Subtasks* are identical and therefore also solved at the same time.

3. **Dependencies** Collaborative *Tasks* can be decomposed into *Subtasks* that have cross-system dependencies. Causal and chronological dependencies between *Subtasks* can thus cross the system boundary.
   Example: In a heterogeneous team of rovers, there is a rover "Scout" that is supposed to geometrically model the environment and a rover "Scientist" that is supposed to collect scientific data about the environment. If the *Task* is to explore the environment scientifically, this can only be done collaboratively, as none of the rovers can solve all *Subtasks*. First, the scout must explore a path to the scientifically relevant areas, then the scientist can drive to these locations and take measurements.
   From the perspective of the *Mission*, there are causal dependencies here that connect the *Subtasks* distributed across the various systems.

4. **Commands** A collaborative *Task* can be split by transferring *Subtasks* across the system boundary via the *Task Interface*. This assigns new *Tasks* to the system. This often also involves cross-border dependencies. Example: A rover and a drone explore an area together. As the drone flies faster but only has a limited flight time, the rover carries the drone most of the time. Each time the rover wants to head for a new destination, it sends the drone ahead to optimize its own path planning. Only when the drone has landed on the rover again are the previously explored points navigated to. The rover therefore determines the *Tasks* of the drone during the *Mission*.
   From the point of view of the *Mission*, the rover hands over *Tasks* to the drone depending on its own *Mission* progress. These *Tasks* then have causal dependencies on the *Tasks* of the rover. This means that *Tasks* are transferred via the interface and cross-border dependencies are created. The superordinate *Task* lies with the commanding rover, the drone only executes *Subtasks* without knowing the superordinate *Task*.

5. **Negotiating** The most complex form of the *Task Interface* is when the distribution of the *Tasks* is negotiated between the *Robots*. This means that there is no fixed hierarchy that defines how the *Tasks* are exchanged between the systems, but based on the *Mission*, the progress and the individual *Skills*, the systems assign the *Tasks* to each other. Example: Exploration with two rovers exchanging exploration strategies. First, the rovers divide up the area. As the

Mission progresses, one rover is faster than the other. This rover then takes over further areas from the other rover.
From a *Mission* perspective, *Tasks* are transferred here via the *Task Interface*. In addition to the assigned *Tasks*, however, all systems have a representation of the higher-level *Task*. Based on this information, the allocation of the *Task* can be renegotiated during execution.

4.7.3 Mission Control Robot

Based on the definition of the *Robot* 3.2 and the definition of *Mission* 3.4, it can generally be said that every *Robot* has a *Mission Control Task*. This *Mission Control Task* addresses all *Tasks* that the *Robot* should execute, as well as all dependencies between them. This *Task* also establishes the connection between the *Mission Environment* and the *Robot Mission*. In abstract terms, the *Mission Control Task* is a collaborative *Task*. The interface to the system is therefore the *Task Interface* of the *Mission Control Task*. The *Mission Control Task* controls and monitors the specification and execution of the *Mission*. Depending on the system, both the *Mission* itself and the associated *Mission Control Interface* can differ considerably. Three dimensions are therefore presented below, which can be used to systematically classify the various *Missions*.

Mission Dynamic

Missions can be classified according to their dynamics. This is a spectrum with fluid transitions. For better understanding, three different reference *Mission* dynamics are introduced as examples, which cover the range of dynamics.

Static *Mission* The *Mission* is static, i.e. no changes to the *Tasks* to be solved occur during runtime. The dependencies between the individual *Tasks* are also static. These can often be mapped as a simple sequence.
Example of “industrial assembly processes”: The steps and the sequence of the individual steps are precisely defined. If all steps are carried out in the defined sequence, the *Mission* is successfully completed. There are very few, if any, variants for solving the *Task*.

Adaptive *Mission* An adaptive *Mission* is a *Mission* that solves one or more *Tasks*. The way in which these *Tasks* can be solved can change during execution due

to internal and external events. It is also possible to change the *Tasks* during execution.
Example "Clear a table": A service robot that has the *Task* to clear a table has an adaptive *Mission*. The *Task* and thus the target state are clearly defined. However, how this *Task* can be solved depends heavily on the start state. Based on this state, it is possible to plan what needs to be done. As a rule, the implementation of the plan leads to the solution of the *Task*. However, unforeseen events, such as emptying the dishwasher, can lead to a necessary adjustment of the *Mission*.

Reactive *Mission* A reactive *Mission* also has *Tasks* that need to be completed. However, the *Tasks* are dynamic and depend heavily on the current situation. Therefore, the system must react to the current situation in order to fulfill the *Mission*. Global planning of the *Tasks* is usually not possible.
Example "soccer robot": A soccer robot has the *Task* to win the game. However, which *Subtasks* must be fulfilled and when depends almost exclusively on the specific situation (e.g. which team is in possession of the ball, current position, position of the opponents, etc.). It is therefore not possible to plan in advance when which *Task* is to be fulfilled. However, strategies can be developed as to how to react in which situation or how certain situations may be created.

Mission Abstraction Level

Missions can be defined at very different levels of abstraction. Here, as well, there is a continuous spectrum of *Mission* abstractions illustrated by three reference abstraction levels: Low-level mission, intermediate-level mission, high-level mission. The abstraction level of the *Mission* depends only partially on the *Task*. This means that the same *Task* can be mapped at different *Mission* abstraction levels. The three reference levels are illustrated below using the example of assembling a piece of furniture.

Low-Level *Mission* In the low-level *Mission*, the *Tasks* are specified in a very specific and fine-grained way. The higher-level *Task*, why a *Subtask* must be fulfilled, is usually not included in the low-level *Mission*.
Example "Assembly in industrial production": The piece of furniture is assembled by programmed movements of automatic machines. The individual *Tasks* of the machines do not contain the higher-level *Task*, but a sequence of movements to be executed. The piece of furniture itself is no longer part of the *Task*,

but only the execution of the trajectories. Due to the lack of information about the higher-level *Task*, no deviation from the sequence is possible.

Intermediate-Level *Mission* The intermediate-level *Mission* specifies how the higher-level *Task* can be divided into smaller steps. However, how these steps are solved is determined by the system, which can solve them in different ways. The system can also adapt the instructions.
Example "Building furniture with instructions": When building an IKEA piece of furniture using the instructions provided, individual work steps are defined. However, the specific execution, e.g. how a screw is inserted into a hole, is not defined. If the work steps are carried out according to the instructions, the overall assembly is successful. Nevertheless, it is possible to deviate from the instructions, as these only represent one solution for successful installation and the relevant information is available to the system.

High-Level *Mission* In a high-level *Mission*, the *Tasks* are defined at a very high level. The system must therefore find a way to solve the *Tasks* itself. The high-level *Mission* often only defines the target state, but not the way to get there. Since the *Task* is not specified in more detail, there are usually very different ways to solve the *Task*. Another challenge of the high-level *Mission* is that the *Task* is often heavily underspecified and therefore the system has to supplement the missing information from various sources.
Example "Assemble the furniture": The high-level task in the IKEA example would be the task "Build the furniture". No instructions or more detailed instructions are provided. The system must therefore independently examine the components and deduce how a piece of furniture can be built.

Missions Phases

Missions can often be divided into different phases. These phases can either run in a fixed order or alternate flexibly. The following phases can be distinguished:

Idle phase In this phase, the *Robot* waits for *Tasks*. For example, all *Tasks* that have been passed to the system may already have been solved. In this phase, the *Robot* also does not perform any monitoring tasks that belong in the execution phase. However, the system is in operation and can receive *Tasks* via the *Mission Control Interface*.

Setup phase In the setup phase, the system is prepared for solving the *Mission*. These can be system-specific *Tasks*, such as the calibration of the sensors. Creating a map or modeling the current environment can also be part of the setup phase. In addition, *Mission*-specific information can also be transmitted to the system, such as the teach-in of configurations.

Execution phase The *Mission* is solved in the execution phase. This applies to both the action-controlled and the perception-controlled *Tasks*. The *Robot* therefore does not have to move continuously during the execution phase, but can also perform observation *Tasks* with static hardware.

In contrast to the other dimensions of the *Mission* classification (dynamics, abstraction), the phases do not form a continuous spectrum, but can be combined largely at will. With the exception of the execution phase, phases can also be completely absent.

4.7.4 Mission Control Interfaces

The *Mission Control Interface* forms the system boundary of the *Mission Control Task*. It connects the *Mission Control Environment* and the *Mission Control Robot*. Technically speaking, this is the *Task Interface* of the *Mission Control Task*, which is a collaborative *Task* between *Robot* and *Mission Environment*. This allows dependencies to be mapped and states to be monitored via the interface. The dependencies can be used to control the *Tasks*. The interface is not always one-directional from *Mission Environment* to *Mission Control Robot*, but rather both directional; some *Mission Control Interfaces* enable the *Mission Control Robot* to modify *Tasks* in the *Mission Environment*.

The *Mission Control Interface* is the interface between *Mission Control Robot* and *Mission Control Environment*. Therefore, the *Mission Control Interface* is strongly influenced by the properties of the *Robot Mission*. For example, the *Mission Control Interface* can be reduced to start and stop for a static *Mission*. Adaptive *Mission* often require a more complex *Mission Control Interface* as even monitoring the process is significantly more complex. If *Tasks* are passed via the *Mission Control Interface*, the degree of abstraction of the *Mission* determines what they look like. In addition, the *Mission Control Interface* can differ in the various phases of the *Mission*. For example, the *Mission Control Interface* can transmit *Tasks* in the setup phase, while in

the execution phase it is limited to monitoring the *Task* states.

The second important factor is the type of *Mission Control Interface*. Since every *Robot* can be controlled from the outside to a certain extent, types 1 (Disjunct *Tasks*) and 2 (Implicit Collaboration) are not applicable as they do not allow control. In the simplest case, the *Mission Control Interface* is therefore an interface of type 3 (Dependencies), which can implement dependencies across the system boundary. For example, the *Mission Environment* can start the execution of *Tasks*.
The next more complex interface type 4 (Commands) also allows new *Tasks* to be transferred via the *Mission Control Interface*. The recipient of the *Tasks* is always the *Robot*. The *Mission Control Interface* therefore allows the *Robot* to transfer new *Tasks* or change existing *Tasks*. However, this does not exclude the possibility that the *Robot* can also change the *Tasks* transferred to it.
An interface of type 5 (Negotiating) has the highest capability. Here, the *Robot* is also able to transfer *Tasks* to the *Mission Environment*. This means that the *Robot* also determines how the *Mission* is solved outside its system boundaries.

4.7.5 Viewpoint Aspects

The *Viewpoint Mission* describes the *Architecture* of the system with regard to the aspect of the *Tasks* and *Missions* that the *Robot* can solve. This also includes the interface to the *Mission Environment*. As the *Tasks* and *Missions* are purely in the physical world, the *Software Concerns* are not relevant for the *Viewpoint Mission*. The *Viewpoint Aspects* are therefore derived from the *Robotic Concerns*. The *Usability* of a system, but also the *Flexibility* are strongly linked to the system interface. The *Viewpoint Mission* addresses this via the *Viewpoint Aspect* Interface.
All *Robotic Concerns* play a role for the *Robot Mission* in a strongly interwoven form. The *Viewpoint Mission* therefore uses the general aspects of the *Mission*: Abstraction, Phases and Dynamic to achieve a separation of concerns.

The four *Viewpoint Aspects* of the *Viewpoint Mission* are presented below.

Abstraction (*AFE 4.4.1*) The *Viewpoint Aspect* Abstraction addresses *Architecture Decisions* regarding the choice of abstraction level. If different abstraction levels have been selected, the relationship between them is documented here.

Phases (*AFE 4.4.2*) The *Viewpoint Aspect* Phases addresses the *Architecture Decisions* regarding the *Mission* phases of an *Architecture*. This documents which phases are used and how they are related.

Dynamic (*AFE 4.4.3*) The *Viewpoint Aspect* Dynamic addresses the *Architecture Decisions* regarding the flexibility of the *Mission*. This is limited to the internal *Mission* of the *Robot*, but is strongly influenced by the *Mission Environment*. These considerations are also part of this *Viewpoint Aspect*.

Interface (*AFE 4.4.4*) The *Viewpoint Aspect* Interface is used to document *Architecture Decisions* regarding the interface to the *Mission Environment*. This includes both the *Mission Control Interface* and the optional *Task Interfaces*.

The *RoAF* therefore defines the *Viewpoint Aspects* (VM-A) for the *Viewpoint Mission*:

VM-A1 Abstraction

VM-A2 Phases

VM-A3 Dynamic

VM-A4 Interface

4.7.6 Model Kinds

The *Model Kinds* then result from the VM-A and the *Model Abstraction Types*. As shown in Table 4.9, the *RoAF* contains 12 *Model Kinds* for the *Viewpoint Mission*. Each *Model Kind* has a defined *Model Abstraction Type* and a defined *Aspect* from the *Viewpoint Aspects*.

	Guideline	Approach	Implementation
VM-A1 Abstraction	VM-M1G	VM-M1A	VM-M1I
VM-A2 Phases	VM-M2G	VM-M2A	VM-M2I
VM-A3 Dynamic	VM-M3G	VM-M3A	VM-M3I
VM-A4 Interfaces	VM-M4G	VM-M4A	VM-M4I

Table 4.9: *Matrix of Viewpoint Mission Model Kinds*

4.7.7 Viewpoint Summary

The *Viewpoint Mission* is used to view the *Architecture* of a *Robot* as a *Tasks* solving agent. In order to work out the relevant aspects, this section analyzes what constitutes a *Mission* in general. As this is closely related to the *Task* concept, this was also described in this section. With the description of the *Mission Control Robot* and the *Mission Control Interface*, the most important elements of a *Robot* from the perspective of the *Mission* were then identified. The three classification dimensions of the *Mission Control Robot* were therefore selected for the *Viewpoint Aspects*. The *Mission Control Interface* is addressed separately via the *Viewpoint Aspect* Interface.
The 12 *Model Kinds* of the *Viewpoint Mission* then result from the combination with the *Model Abstraction Types*.

4.8 Correspondence Rules

The *Viewpoints* define perspectives in order to view the entire system with respect to defined aspects. The *Views* are therefore self-contained. There are therefore no classic interfaces, such as between modules or layers. Nevertheless, there are correspondences between the various *Views* and their elements if entities are visible from more than one *Viewpoint*. To document this an *Architecture Framework* defines so called *Correspondence Rules*(*AFE 5*). The *Correspondence Rules* identified by the *RoAF* are shown in Figure 4.15.

Hardware abstraction (*AFE 5.1*) The hardware components of the *Robot* can be seen from the *Viewpoint Physical*. These components are accessible via software. From the perspective of the physical world, there are therefore software interfaces. These software interfaces correspond to the hardware interfaces of the *Viewpoint Capabilities*. This is a 1 : 1 relationship.

Capability trigger (*AFE 5.2*) *Skills* have *Capabilities* as components. These correspond to the *Capabilities* of the *Viewpoint Capabilities*. A *Skill* can contain many *Capabilities* as components; conversely, *Capabilities* can be used in various *Skills*.

Task solving (*AFE 5.3*) *Skills* solve *Tasks*: Since each *Skill* is designed to solve *Tasks*, a *Correspondence* can be established between the *Viewpoint Mission* and the *Viewpoint Skills* for *Tasks* that are solved by *Skills*. *Skills* are designed in such a

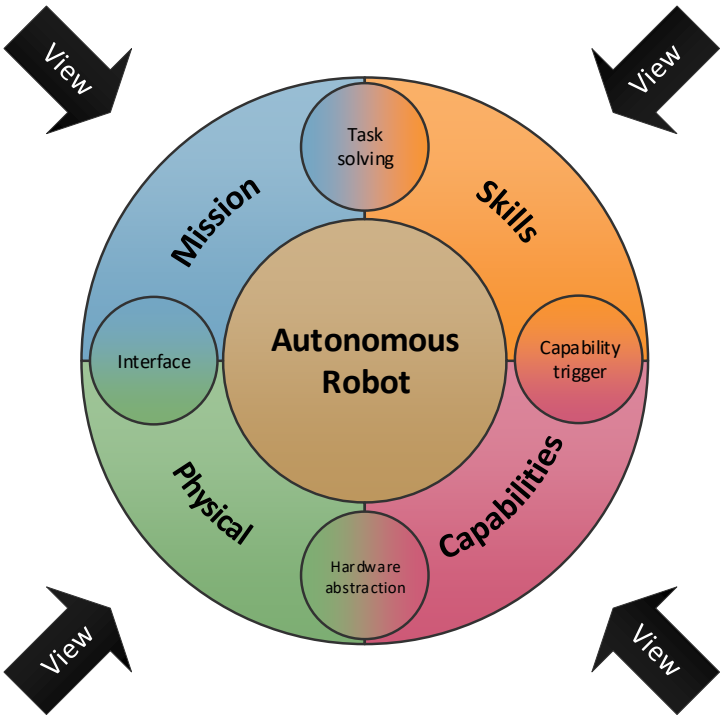

Figure 4.15: *If elements of an architecture are visible from at least two viewpoints, Correspondences occurs. The Correspondence Rules document the occurrence of these Correspondences. For the RoAF, four Correspondence Rules are identified as shown. These are Hardware abstraction, Capability trigger, Task solving and Interface.*

way that they can be applied to different *Tasks*. Conversely, a certain *Task* can also potentially be solved by different *Skills*.

Interface (*AFE 5.4*) *Robots* are physical systems. Each interface therefore requires a physical equivalent. These can be dedicated concepts that are described in the *Viewpoint Aspect* Interfaces of the *Viewpoint Physical*, but also interfaces that are implemented by IT components.

These listed *Correspondence Rules* result from the definition of the *RoAF*. They are therefore generally valid and do not need to be included in the *Architecture Description* of a *Robot*.

4.9 Chapter Summary

In this chapter, the *RoAF*, an *Architecture Framework* for the domain of robotics, was developed and its components presented. In Figure 4.16 an overview of the identified components is depicted. This included the identification of the *Stakeholders*, which classifies the *RoAF* into the groups *User* and *Developer*. The general *Concerns* of the domain and their relationships to each other were then described. They are subdivided in the *Robotic Concerns* and the *Software Concerns*.
Since the *Concern Complexity* is of great importance in robotics and has relationships to all other *Concerns*, the *RoAF* was extended by the construct of the *Model Abstraction Types*, depicted in Figure 4.16 as purple circles. These make it possible to view the *Architecture* at different levels of abstraction.
Despite this extension, the *RoAF* remains an ISO 42010-compatible *Architecture Framework*.
Based on this, the four *Viewpoints* were presented, which form the core of the *RoAF*. These divide the *Architecture Description* into the perspectives *Physical*, *Capabilities*, *Skills* and *Mission*. Finally, the relationships between the *Viewpoints* are described via the *Correspondence Rules*. Including the *Model Kinds* of the *RoAF* a total of 121 *Architecture Framework Elements* are defined.

All these *AFEs* are *Architecture* agnostic but define the conventions how to describe a *Robot Architecture*. In the next chapter, an explicit process to create *Architecture Descriptions* on the basis of the *RoAF* is presented.

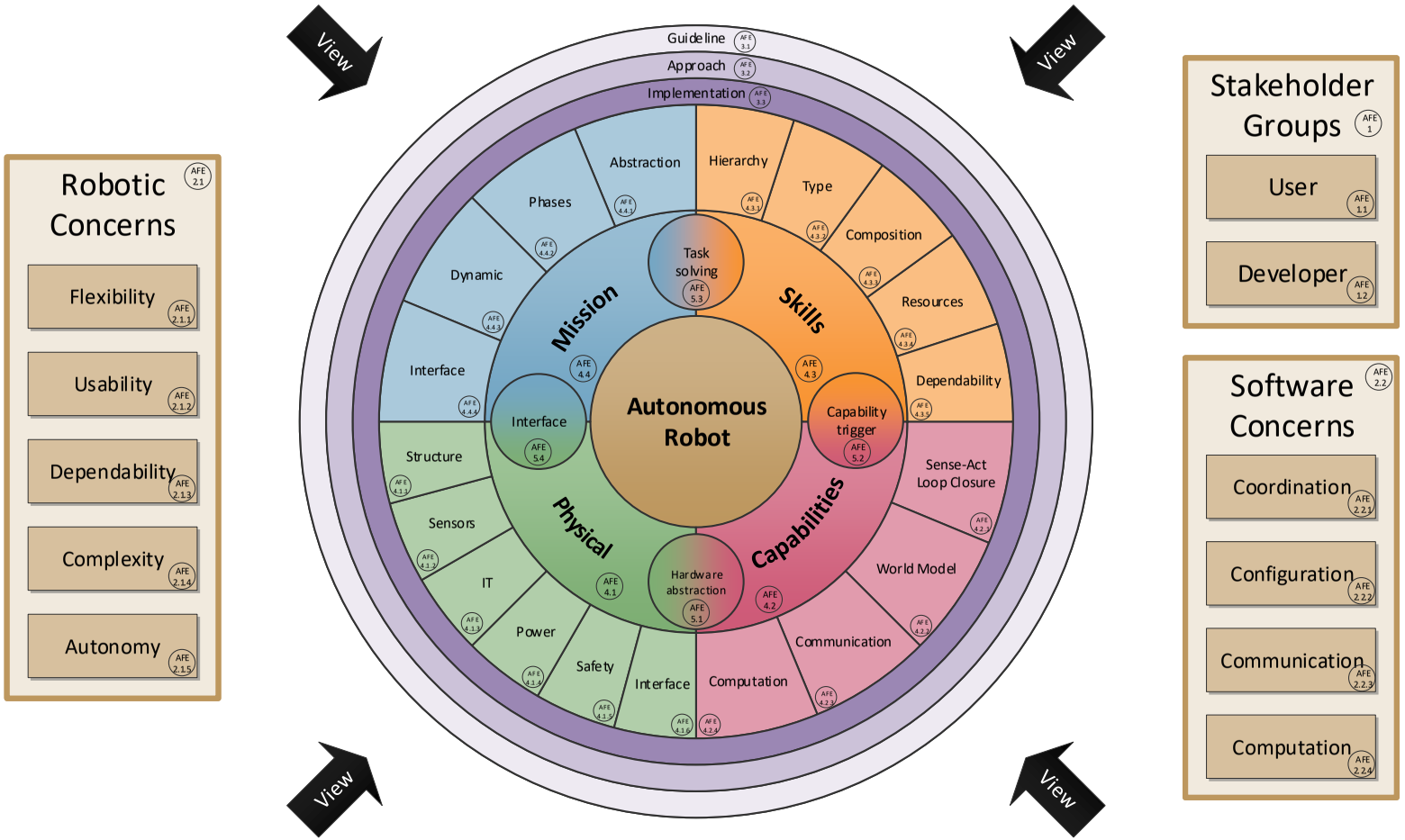

Figure 4.16: *The RoAF consists of a variety of AFE components. In the brown boxes, excerpts from the elements of the Architecture Context, consisting of Stakeholders (AFE 1) and Concern (AFE 2), are shown. The actual Architecture description, i.e. the documentation of the Architecture Decisions is realized by the Viewpoints (AFE 3) Each Viewpoint is subdivided by Viewpoint specific Aspects. Orthogonal to the subdivision by the Viewpoints, the RoAF subdivides the description by the Model Abstraction Types (AFE 4) represented by the purple rings. Overlaps between the Viewpoints are documented in the Correspondence Rules (AFE 5), shown as circles between the Viewpoint.*

Chapter five

Robot Architecture Description Process

This chapter presents the developed process for creating an *Architecture Description* based on the *RoAF*. As shown in Figure 5.1 the input to this process is a *Robot* with

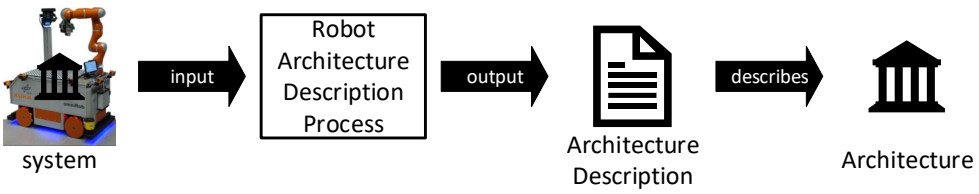

Figure 5.1: *The Robot Architecture Description Process has a system as input and extracts the contained Architecture by creating an Architecture Description.*

its contained *Architecture*. The result of the process is a *Architecture Description* which expresses the system's *Architecture*. The process therefore identifies the most important concepts and design decisions and documents them in the conventions specified by the *RoAF*.

All elements that are involved in that process are referred to as *Architecture Description Elements*, in accordance with ISO4010. In Figure 5.2 an overview of the *AD Elements* is shown. The elements are divided into two groups.

One group are the *Architecture Framework Elements* outlined in black in Figure 5.2, which were presented in the previous chapter Chapter 4. These are *Architecture*

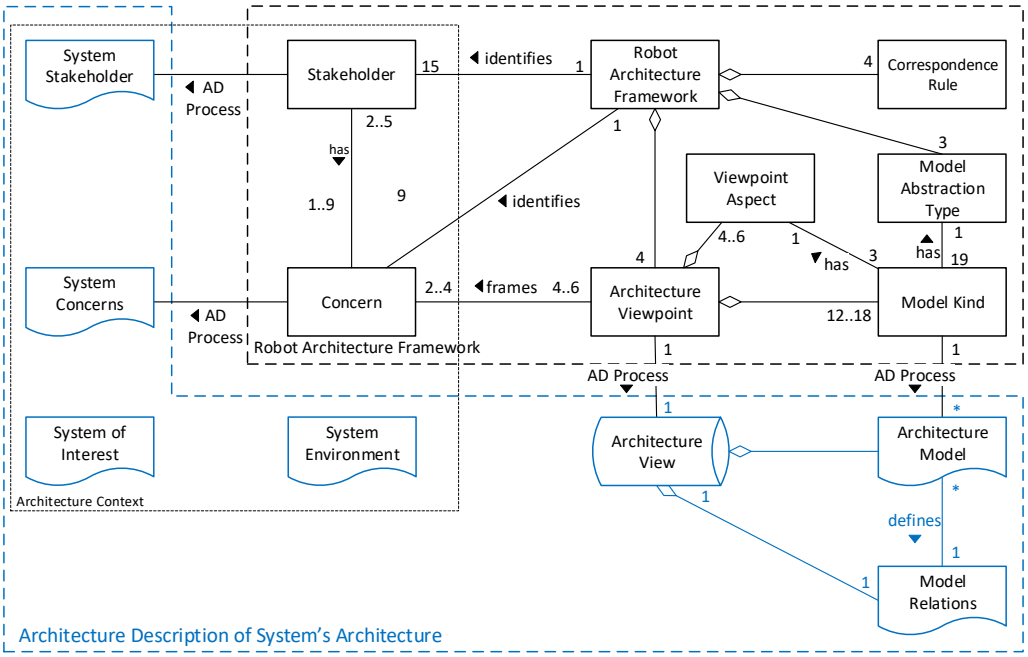

Figure 5.2: *The Robot Architecture Description Process creates Architecture Descriptions (blue frame) from Robot Architectures on the basis of Architecture Framework Elements (black frame). A Architecture Description consist of the system-specific Architecture Context and the different Views of the system Architecture, which are derived from the Viewpoints by the Architecture Description Process. Each View consists of the Architecture Models that document Architecture Decisions based on the Model Kinds. (adapted from ISO42010 [59])*

agnostic and define the conventions for describing *Architectures* in the robotics domain.

The *Robot Architecture Description Process* then uses this as a basis to create the second group of *AD Elements*, outlined in blue in Figure 5.2, which represents a specific *Robot Architecture* as an *Architecture Description*.

Based on the *AFE Stakeholders* and *Concerns*, the *Architecture Context* of the *Robot* is determined. For each *Viewpoint*, a *View* is created by using the *Model Kinds* to document individual *Architecture Decision* in *Architecture Models*. The relationships between the *Models* of a *View* are each documented as relations model. Together, these elements provide a complete *Architecture Description*.

To determine the *Architecture Description* the *Robot Architecture Description Process* is divided into various independent sub-processes, as shown in Figure 5.3.

A complete *Architecture Description* consists of a description of the *Architecture Context* and four *Views* on the *Architecture* from the perspective of the four *Viewpoints* defined in the *RoAF*: *Viewpoint Physical*, *Viewpoint Capabilities*, *Viewpoint Skills* and *Viewpoint Mission*. As these processes are independent of each other, it is also possible to create

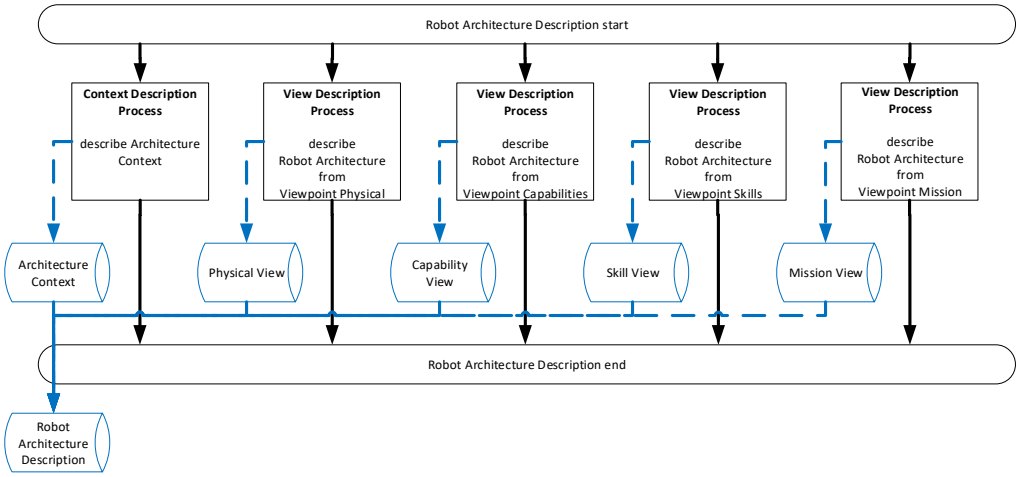

Figure 5.3: *Process of describing a Robot Architecture*

partial *Architecture Descriptions* that only include one *View*, for example. It is also possible to distribute the *Architecture Description Processes* to different people. For example, the hardware developer can describe the *Architecture* from the perspective of the *Viewpoint Physical*, while a software developer creates the *Capability View*. The various sub-processes are presented below.

5.1 Architecture Context Description Process

The first sub-process of the *Architecture Description Process* is the description of the *Architecture Context*. No *Architecture Decisions* are presented here, but the context in which the *Architecture* is used is described. As this is essential for understanding the *Architecture Decisions*, the *Architecture Context* is an important part of the *Architecture Description*.

In Figure 5.4 the process of the *Architecture Context* description is shown. It is a sequential process of four sub-processes, each of which describes an aspect of the *Architecture Context*. Since the *RoAF* is used to describe *Architectures* of concrete systems, the most important context of an *Architecture Description* is the corresponding system. The *Architecture Context* therefore contains a reference to the corresponding *Robot*. This consists of a brief description of the system and references to publications about the system.
The second element is a description of the system's environment. As described in

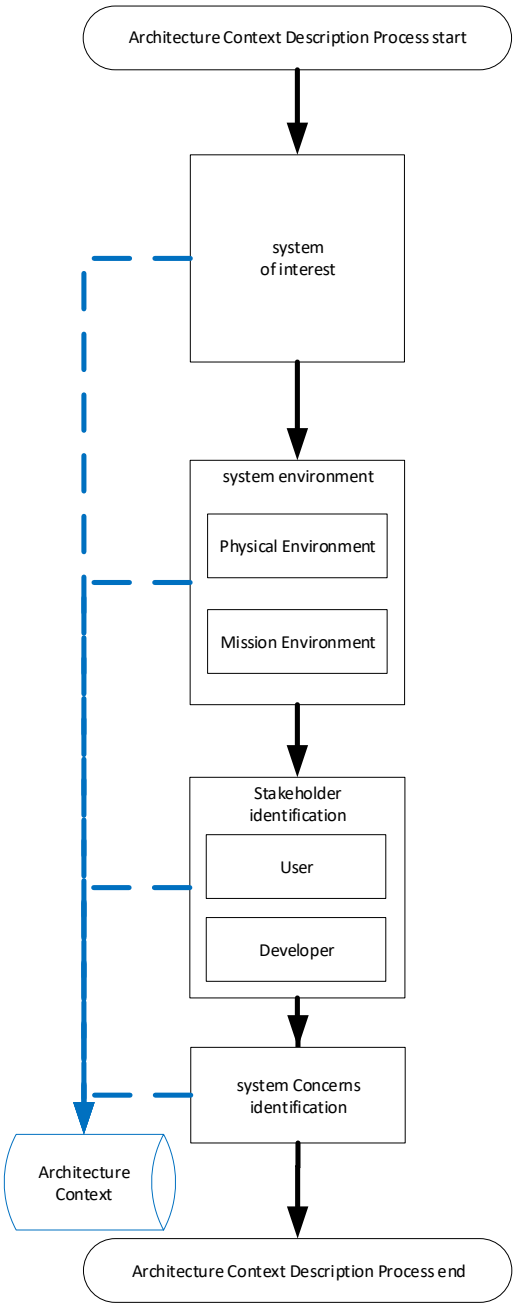

Figure 5.4: *Process of describing the Architecture Context of a Robot*

Subsection 3.2.1, each *Robot* has two types of system boundaries. This results in two environments: the *Physical Environment* and the *Mission Environment*. Both are therefore captured by the process in the *Architecture Context*.
The next element is the identification and description of the *Stakeholders* of a system. As described in Section 4.1, *Stakeholder* are people who have an interest in the system. On the one hand, these are the *Users* of the system that use the *Robot* to solve *Tasks* with it, but also the *Developer* of the system that design and implement the *Robot*. Even though the *Stakeholder* do not explicitly affect the *Architecture* of a *Robot* it helps to understand the context of an *Architecture*. A *Robot* designed by mechanical engineers has usually a different focus than a *Robot* designed by AI researcher.
The last element is the description of the system's *Concerns*. These are derived from the *Concerns* of the *RoAF*, but have system-specific weightings. For example a space rover will have a higher focus on *Dependability* than a soccer robot. It is therefore important context information how the *Concerns* are prioritized.

5.2 View Description Process

The creation of *Views* is the central element of the *RoAF*. All *Architecture Decisions* made are documented here. For a complete *Architecture Description*, a system must be described from all *Viewpoints* specified in the *RoAF*. This process runs analogously for each *Viewpoint* and is therefore described generically in this section.

As shown in Figure 5.5, the process of *View* creation generally consists of four different modeling processes. These processes are arranged sequentially but also allow iterative modeling through feedback. The three *Architecture Description Processes* can in turn be generalized and are presented in detail in the next section 5.3. The initial process when creating an *Architecture View* is the modeling of the *Guideline Models* on the basis of the *Viewpoint*. Once this process is complete, the *Approach Models* of the *Viewpoint* are generated. Since *Approach Models* are linked to the *Guideline Models*, these are referenced. It may be the situation that *Guidelines* are identified that have not yet been modeled. If this is the case, the *Guideline* process is restarted. After modeling the *Approaches*, the process of modelling the *Implementation Models* is started. These in turn refer to the *Approach Models*, which means that any gaps can also be uncovered here. The process therefore allows feedback on the *Approach* modeling. Finally, the *Relations Model* is created. This documents the dependencies between the *Models* of the *View*, but does not contain any new information itself, as

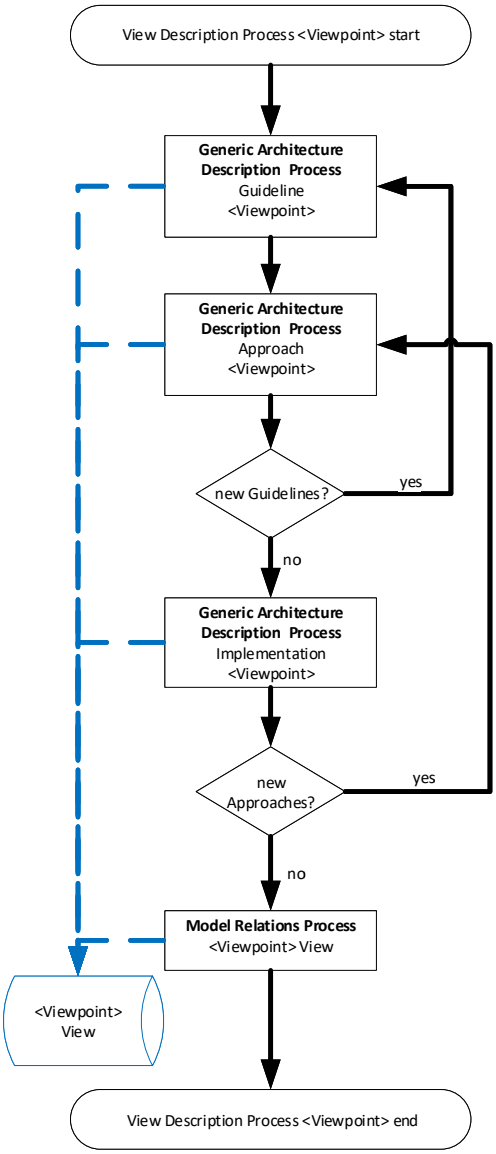

Figure 5.5: *A View is created by describing the Architecture from a particular Viewpoint. The RoAF uses the levels of abstraction defined by the Model Abstraction Types. Therefore a View of the RoAF contains an additional separation into the three abstraction levels. First, the Architecture is described at the Guideline level. Then a description is given at the Approach level. Due to the relationships between the levels of abstraction, this may reveal new Architecture Decision at the Guideline level, which is represented by the loop back. Then the Implementation level is addressed, including a loop back to the Approach level. Finally, the relationships between the Models of the different abstraction levels are documented in a relation model.*

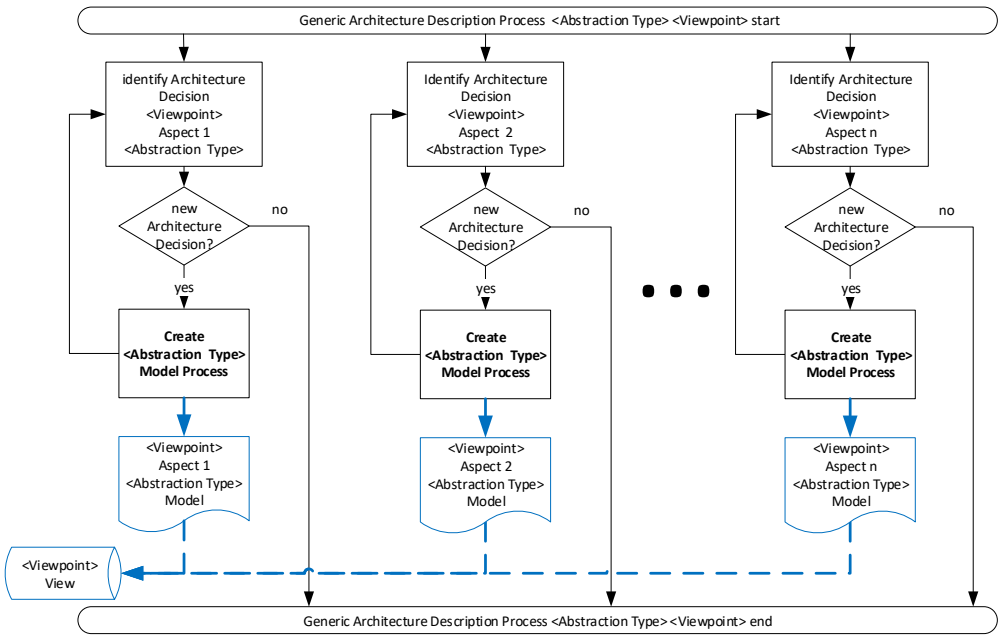

Figure 5.6: *Generic template for Architecture Description Process*

this is already recorded in the individual *Models*. The result of the overall process is the *View* on the system from the specified *Viewpoint*.

5.3 Generic Architecture Description Process

To create a *View*, one *Architecture Description Process* is started for each abstraction level of the *RoAF*, i.e. for the three *Model Abstraction Types Guideline*, *Approach* and *Implementation*, an *Architecture Description Process* is started. As this process runs in the same way for the different *Viewpoints* and for the different abstraction levels, it is described and depicted in Figure 5.6 in generic form.

As shown, the system is analyzed with regard to all *Viewpoint Aspects* defined in the *Viewpoint*. The identified *Architecture Decisions* are then documented in the form of *Architecture Models* for the *Architecture Description*. Based on the defined *Model Abstraction Type* and the *Viewpoint Aspect* the appropriate *Model Kind* is selected. Any number of *Models* can be created for each *Viewpoint Aspect*, all of which are added to the *View* of the *Architecture Description*. Therefore a loop back in the process iterates until all *Architecture Decision* relevant to the *Viewpoint Aspect* are identified.

As can be seen in the figure, the respective *Viewpoint Aspects* are considered in parallel and also define the *Model Kind* of the created *Models*. However, a single *Model* can address additional *Viewpoint Aspects*, as explained in the next section.

5.4 Model Creation Processes

A Model Creation Process creates a *Architecture Model* based on a specified *Model Kind*. Since there are deviations in the modeling process depending on the *Model Abstraction Type* of the *Model Kinds* a separate *Model* creation process is presented for each abstraction level.

5.4.1 Create Guideline Model Process

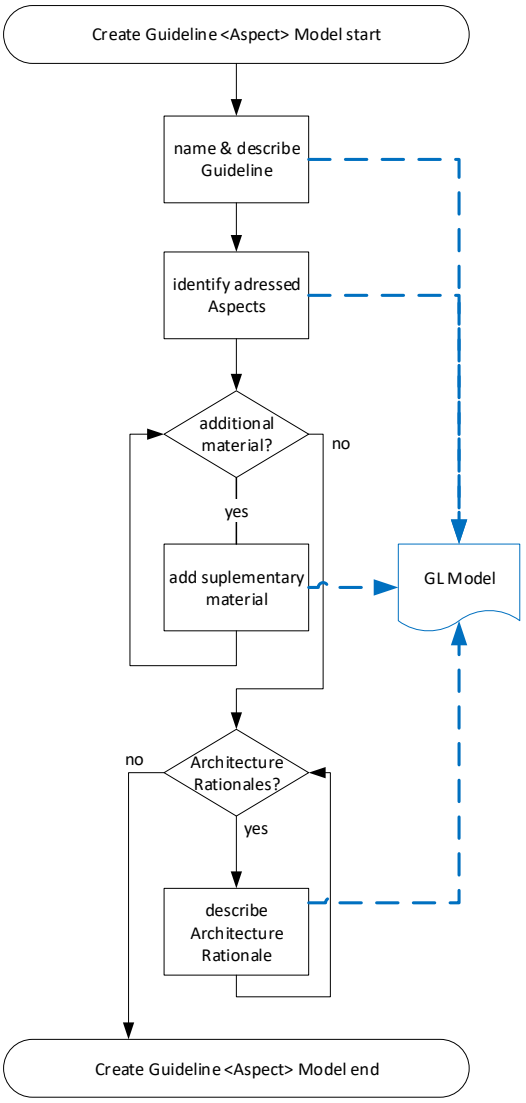

Figure 5.7: *Process of creating Guideline Models*

The process of creating a *Guideline Model* is shown in Figure 5.7. The first step is to name and describe the *Architecture Decision* and identify the addressed *Viewpoint Aspects*. The process defines the main *Aspect* of an *Architecture Decision*, but any number of other *Viewpoint Aspects* can be addressed. This is the mandatory part of each *RoAF*'s *Architecture Model*. In the next step, further *AD Elements* can be added. This is an optional step and can include graphics, diagrams and more detailed descriptions, for example. The last part consists of determining the *Architecture Rationales*. Here it can be described why a certain decision was made. Alternative *Architecture Decisions* can also be discussed here.

5.4.2 Create Approach Models

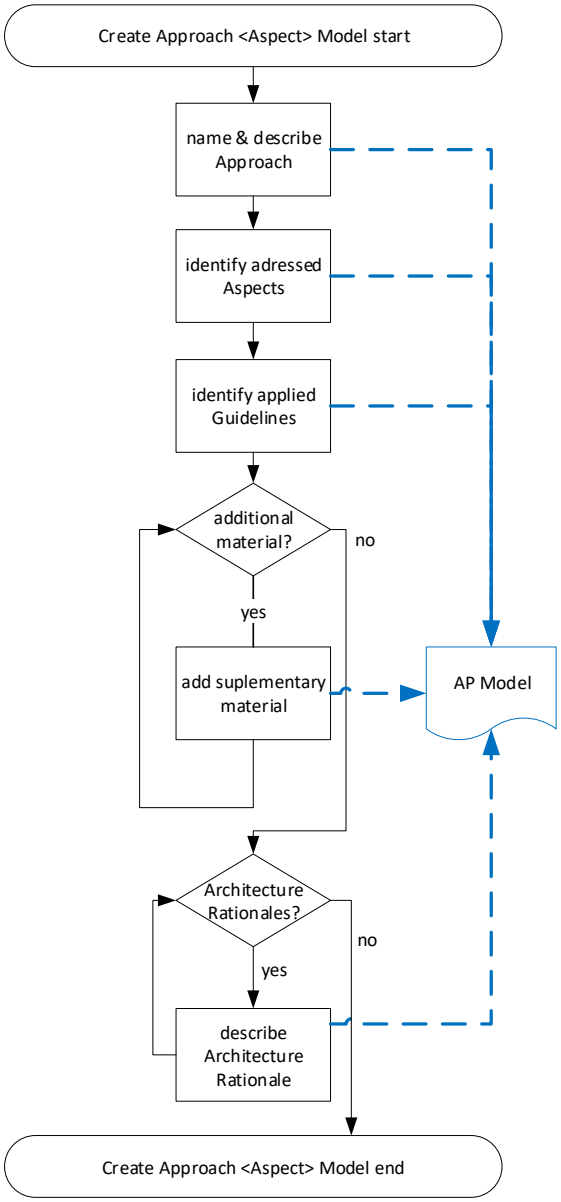

Figure 5.8: *Process of creating Approach Models*

The process shown in Figure 5.8 for creating a *Approach Model* includes the same steps as the creation of a *Guideline Model*. However, in addition to these steps, the identification of the *Guidelines* used is included. This links the *Guideline Models* with the *Approach Models*. Each *Approach Model* must apply at least one *Guideline Model*. This can also include *Guideline Models* that address other *Viewpoint Aspects*.

5.4.3 Create Implementation Model Process

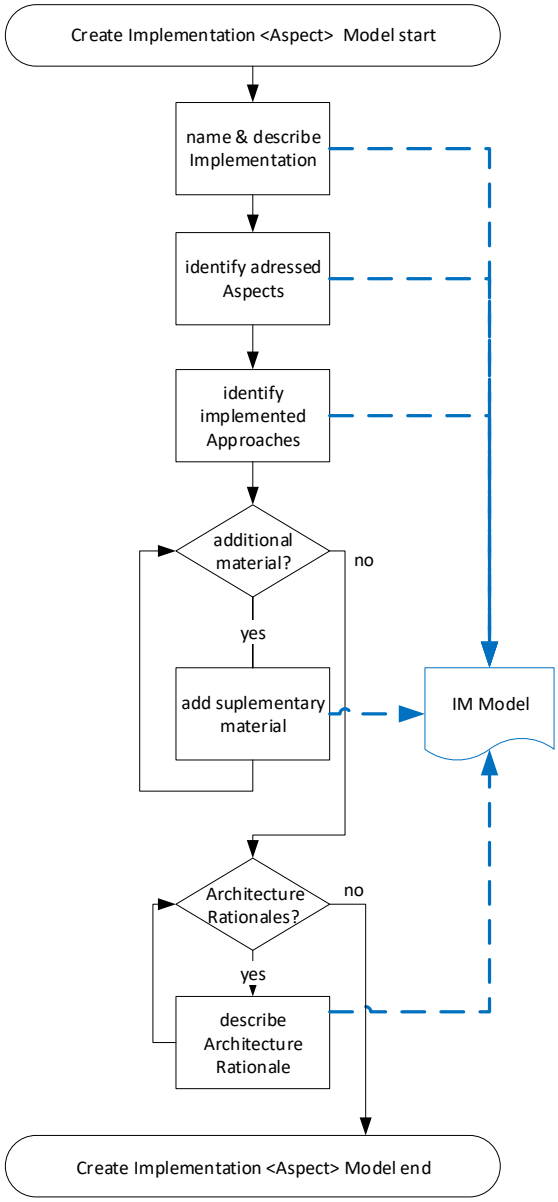

Figure 5.9: *Process of creating Implementation Models*

The modeling process of a *Implementation Model*, as shown in Figure 5.9, is structured analogously to the *Approach* modeling process. A *Implementation Model* must reference at least one *Approach* model, but can also reference several *Approach Models* from different *Viewpoint Aspects*.

5.4.4 Create Relations Model Process

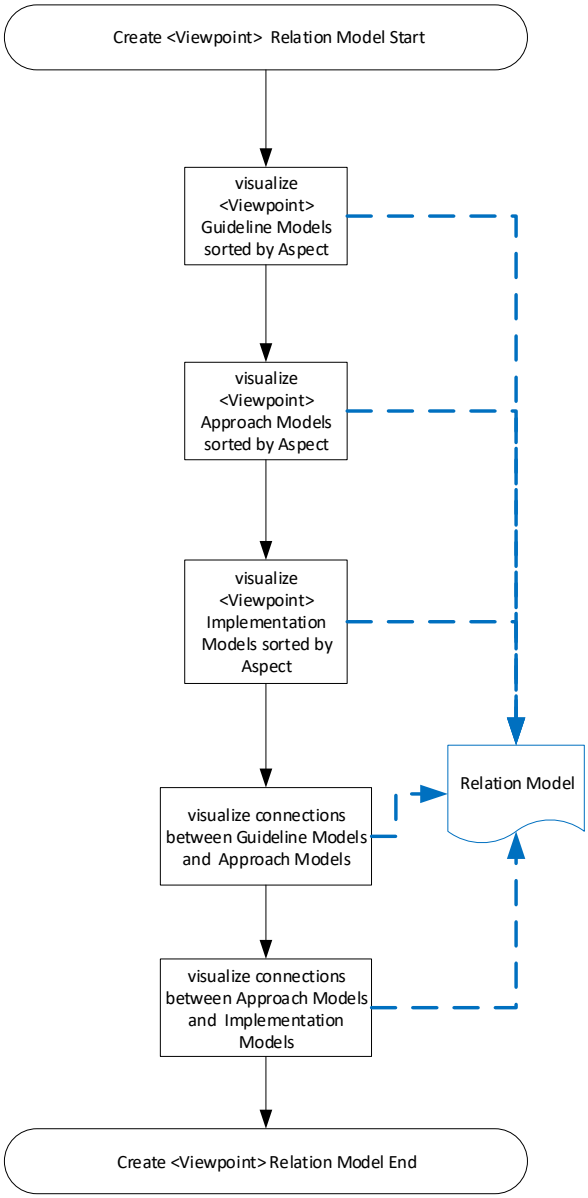

Figure 5.10: *Process of creating the Relations Model*

Each *View* contains a *Relations Model*. This is a metamodel [59]. This means that no new information is presented, but the relationships defined in the individual *Models* are displayed compactly in a graphic. This makes it possible to get a quick overview of a *View*. In particular, the visualization across the *Viewpoint Aspect* boundaries helps to understand the relationships. This process is displayed in Figure 5.10.

5.5 Full Process of Creating an Architecture Description

The creation of a complete *Architecture Description* based on the *RoAF* is composed of the parameterized processes presented in the previous sections. The *RoAF* thus defines a overall process based on the *Viewpoints* and *Viewpoint Aspects* contained in it and the processes defined in this chapter. As shown in 5.11, this consists of the description of the *Architecture Context* and the description of the four *Views*.

If this process is applied to a *Robot*, the result is a systematic description of the *Architecture*. The parallelization and generalization of the process and the individual sub-processes result in several advantages. For example, it is possible to reduce the *Architecture Description* to individual *Viewpoints*. The resulting description then only contains the selected *Views*. Aspects that are not covered by these are not documented. However, due to the independence of the individual *Views*, they describe the system's *Architecture* completely with regard to the respective perspective.

Another advantage of generalization is the simple expandability of the process. If it is necessary to add further *Viewpoints* to the *RoAF*, this is possible directly. Existing *Architecture Descriptions* are not invalidated by this, but must be extended by the corresponding *View* for a complete *Architecture Description*.

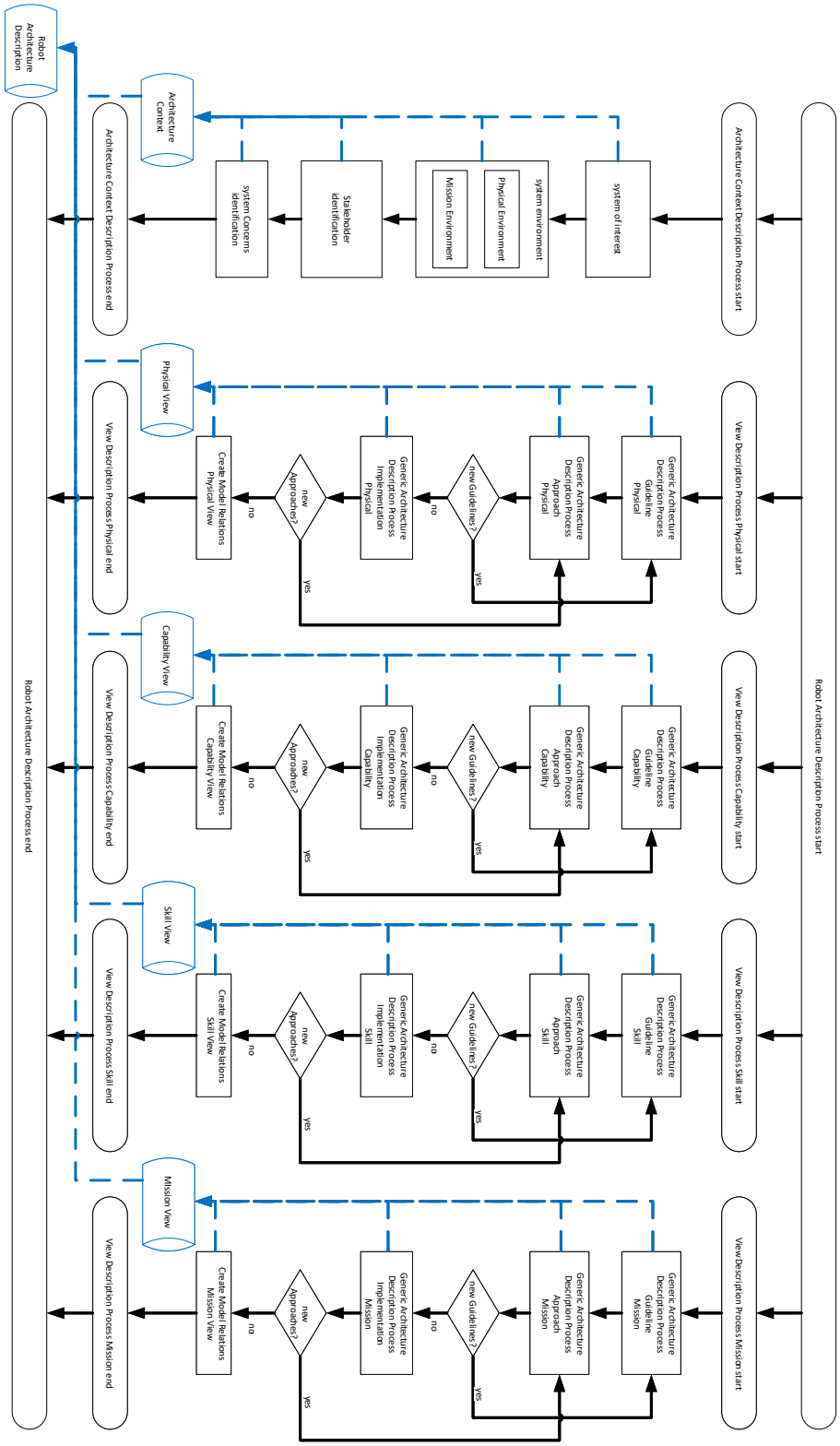

Figure 5.11: *Full Architecture Description Process of the RoAF*

5.6 Chapter Summary

In this chapter, an explicit process was defined on the basis of the *RoAF*, which generates a *Architecture Description* when applied to a *Robot*. When creating the process, attention was paid to a modular structure. A complete *Architecture Description* results from the creation of the *Architecture Context* and the four *Views*. These sub-processes have no dependencies on each other, so that the *Architecture Description* can be reduced to a subset of *Views*.
First, the process for describing the *Architecture Context* was described. The process for describing a *View* was then presented. As this runs analogously for each *Viewpoint*, it was described in generic form. In this process, the sub-processes for the different levels of abstraction run analogously. This sub-process was therefore also described generically. The creation of a *Model* then represents the finest granularity of the process. As this differs for the various *Model Abstraction Types*, these were described separately. Finally, the overall process that results for the *RoAF* from the combination of the various process modules was described. In the next chapter, the *RoAF* is verified and validated exemplary but also conceptually.

Chapter six

Verification and Validation of the Robot Architecture Framework

With the *Robot Architecture Description Process* presented in the last chapter, it is possible to create *Architecture Descriptions* of complex *Robots* based on the *RoAF*. This chapter verifies that the requirements defined in Section 1.2 are met: The process must be applicable to various types of systems, the generated *Architecture Descriptions* must be compact, and it must be possible to describe systems in partial aspects as well. It is also validated that the goals of comparing complex systems, evolving existing systems, and supporting the design of new systems are achieved. Verification and validation are carried out both exemplary and conceptually. For the exemplary validation, *Architecture Descriptions* of three very different systems were created.

The chapter therefore begins with the presentation of the three *Architecture Descriptions*. Based on these descriptions and conceptual considerations, the applicability of the *RoAF* is first verified. Then the *RoAF* is validated in terms of its utility.

Figure 6.1: *The autonomous industrial mobile manipulator AIMM*

6.1 Application of the Robot Architecture Framework to various Robots

The *Architectures* of three very different systems are described below. First, a complete *Architecture Description* of an industrial mobile manipulator is presented. Then the *Mission View* of a planetary exploration rover is given. Finally, the *Physical View* of an autonomous drone is described.

6.1.1 Architecture Description of the Mobile Manipulator AIMM

The AIMM system is a research platform for researching and developing mobile manipulation in an industrial context. The focus is on the software components of the *Robot*, in particular the methodology for environment perception and manipulation planning. The aim of the development is to enable its use in industry-related scenarios without the need for robotics experts for system integration. The system must therefore include the capabilities of a system integrator. *Autonomy* is therefore a main focus of research on the AIMM system. The *Robot* should be able to take on new *Tasks* independently and react appropriately to its environment. The system

	System	Environment	*Stakeholders*	*Concerns*	Total
word count	226	164	826	684	1900

Table 6.1: *AIMM Architecture Context*

was developed as part of the EU TAPAS project and has since been used in various research projects.

As part of this work, a complete *Architecture Description* of the AIMM system with the *RoAF* was created. This consists of the *Architecture Context* and the four *Views* that can be generated from the *Viewpoints* of the *RoAF*.

Architecture Context The *Architecture Context* is used to classify the *Architecture Description*. The full description of the *Architecture Context* can be found in the appendix A of this thesis.

In order to quantify the extent of this description, similar to publication formats, the number of words is used, see Table 6.1. For the description of the AIMM *Architecture Context* consisting of the system, its environment as well as the *Stakeholders* and the system-specific *Concerns*, 1900 words were used.

Physical View The *Physical View* describes the *Architecture* of the system from the *Viewpoint Physical*. The complete *Physical View* of the AIMM system can be found in the appendix B.

In Figure 6.2, the *Relations Model* of the *View* is displayed. It shows all *Models* and their relationships with each other. As shown in Table 6.2, the *View* consists of 27 *Models*. These include 11 *Guideline Models*, 8 *Approach Models* and 8 *Implementation Models*.
The *Models* in the *Viewpoint Aspects* Power, Safety and Interface have no relationships to *Models* of other *Viewpoint Aspects*. In addition, these *Models* each form a chain consisting of a *Guideline*, an *Approach* and an *Implementation Model*. They thus form the simplest structure of the *RoAF*. One *Approach Model* of the *Viewpoint Aspect* Interface is not addressed in any of the *Implementation Models* presented.
The *Models* of the *Viewpoint Aspects* IT, Perception and Structure have connections across the *Aspects*. This models the dependency between these *Viewpoint Aspects* in the *Architecture* of the AIMM system. The structure of the *Models* is also much more complex. For example, many *Models* are referenced by several *Models* of the lower abstraction level. Similarly, some *Models* are based on several *Models* of the higher abstraction level. The various *Guidelines* and *Approaches* are therefore not independent of each other in the AIMM *Architecture*.

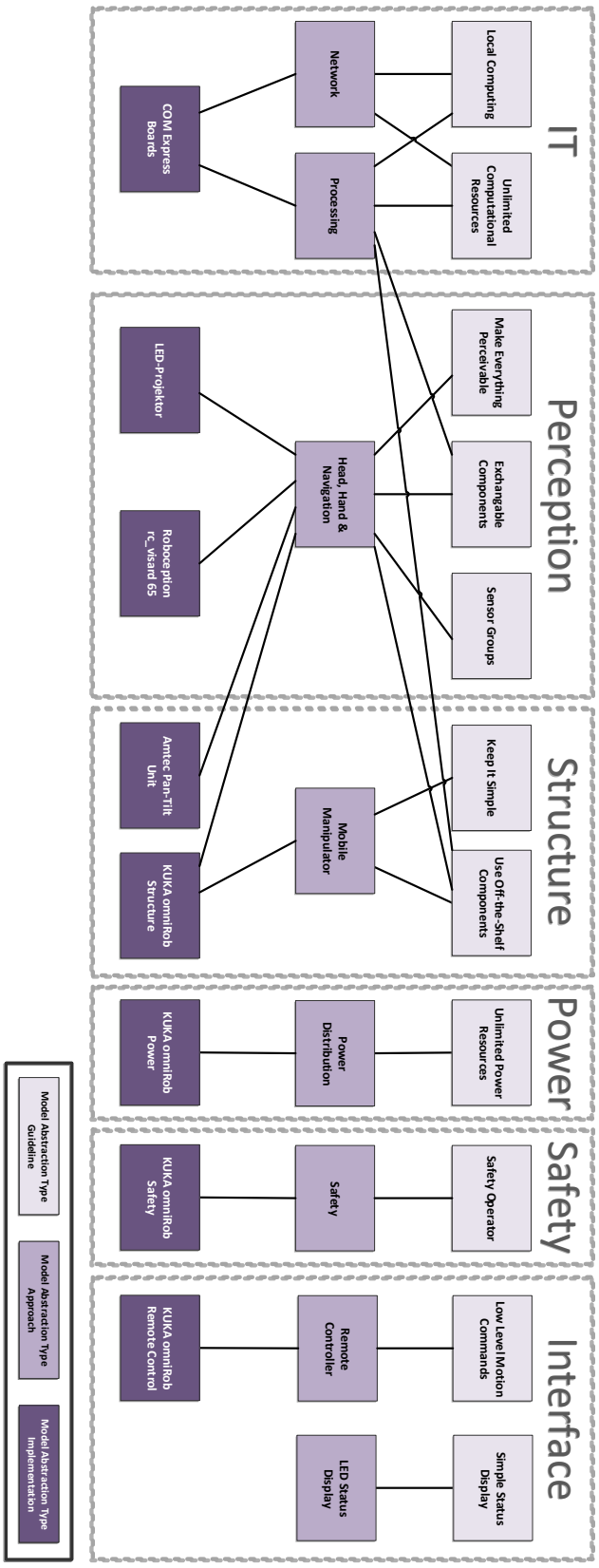

Figure 6.2: *Relations Model AIMM Physical View*

	Guideline	*Approach*	*Implementation*	total
Models	11	8	8	27
Rationales	32	20	6	58
word count	2565	2684	996	6245

Table 6.2: *AIMM Physical View*

A total of 58 *Architecture Rationales* were documented in this *View*. With 32 *Rationales*, most *Architecture Decisions* were described in the *Guideline Models*. 20 *Rationales* are attributable to the *Approach Models* and only 6 to the 8 *Implementation Models*. A total of 6245 words were used to describe all *Models* and the associated *Rationales*. Of these, the largest proportion, with 2684 words, is accounted for *Approach Models*. The *Implementation Models* are described with 996 words. On average, the *Approach Models* are therefore almost three times as large as the *Implementation Models*.

Capability View The *Capability View* of the AIMM system is also described in full in the appendix C. The *Relations Model* of the *View* is shown in Figure 6.3. Here, the *Architecture Models* of the *View* are grouped according to the *Viewpoint Aspects*. The *View* consists of 23 *Models* including 12 *Guideline Models*, 7 *Approach Models* and 4 *Implementation Models*.

The *Models* of the various *Viewpoint Aspects* are relatively independent in this *View*. Only one *Implementation Model* establishes a connection between the *Aspects* Communication, Computation and Sense-Act Loop. However, the structure within the *Aspects* is complex and highly interconnected. This shows that the *Viewpoint Aspects* provide a good separation for the *Architecture* of the AIMM system.

	Guideline	*Approach*	*Implementation*	total
Models	12	7	4	23
Rationales	38	19	8	65
word count	8203	4598	1130	13931

Table 6.3: *AIMM Capability View*

The *Capability View* documents a total of 65 *Rationales*, as shown in 6.3. The majority of the *Rationales*, 38 *Rationales*, are accounted for the *Guideline Models*. 19 *Rationales* justify the *Approach Models* and 8 *Rationales* the *Implementation Models*. This is also reflected in the number of words used. The *Guideline Models* use 8203 words, the

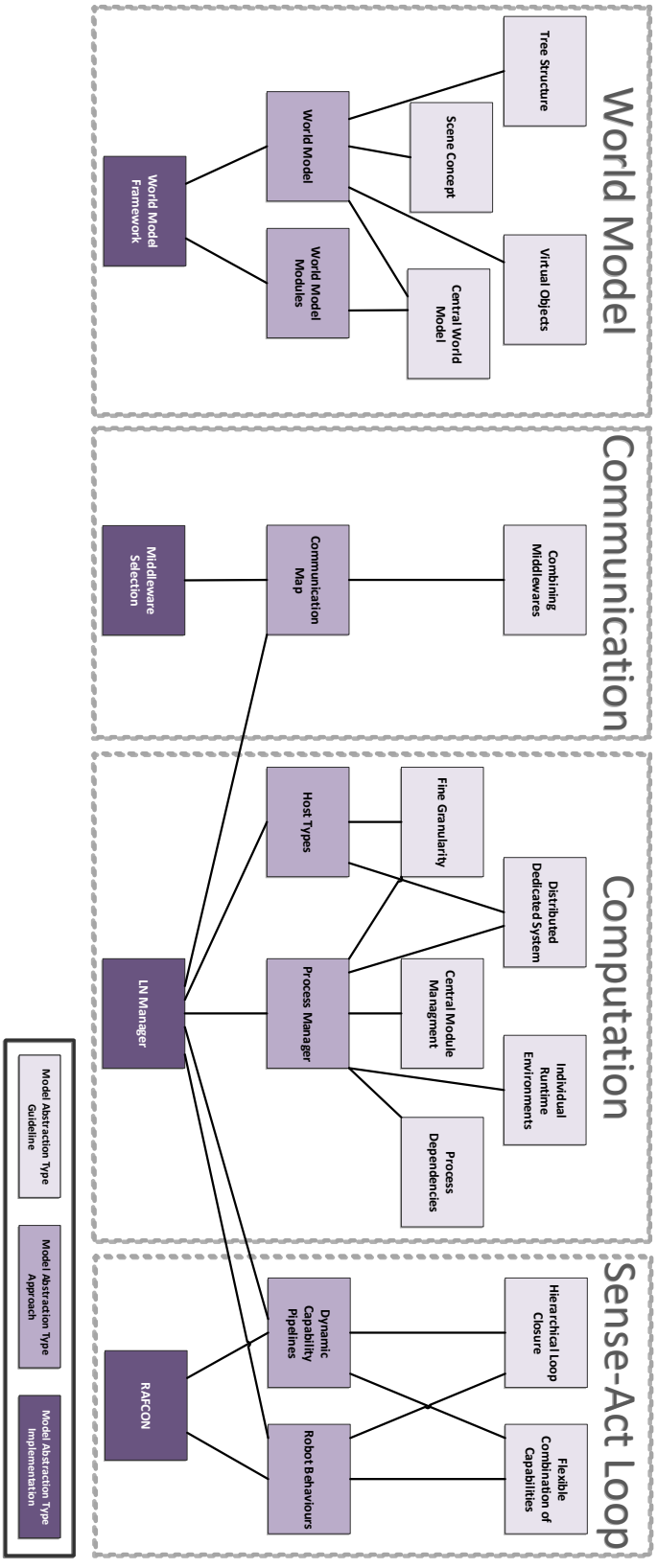

Figure 6.3: *Relations Models AIMM Capability View*

Approach Models 4598 words. The four *Implementation Models* are described with 1130 words. In total, the *Capability View* of the AIMM system is described with 13931 words.

Skill View The *Skill View* describes the *Architecture* of the system from the *Viewpoint Skills*. The full *Skill View* of the AIMM system can be found in the appendix D.

In Figure 6.4, the *Relations Model* of the *View* is displayed. As shown in Table 6.4, the *View* consists of 23 *Models*. This includes 12 *Guideline Models*, 7 *Approach Models* and 4 *Implementation Models*.
The *Models* of the *Viewpoint Aspects* Resources and Dependability only have relationships within their respective *Viewpoint Aspect*. Within the *Aspects*, however, they are strongly interconnected.
The *Models* of the *Viewpoint Aspects* Type, Hierarchy and Composition, on the other hand, are also strongly interlinked.

	Guideline	*Approach*	*Implementation*	total
Models	12	7	4	23
Rationales	37	20	0	57
word count	4904	3077	472	8453

Table 6.4: *AIMM Skill View*

The *Skill View* documents a total of 57 *Rationales*. The four *Implementation Models* of the *View* do not contain any *Rationales*. Most *Rationales* can be found, like in all *View* of the AIMM system, in the *Guideline Models*. There are 37 *Rationales* described here. The *Approach Models* are justified with 20 *Rationales*.
Overall, the *Skill View* is described with 8453 words. More than half of these words are accounted for *Guideline Models* with 4904 words. The *Approach Models* accounts for 3077 words. On average, the individual *Guideline Models* and the *Approach Models* are each described with approx. 400 words. The 4 *Implementation Models* of the *Skill View*, however, are described together with 472 words. They are therefore significantly smaller than the *Models* of the other abstraction levels with approx. 100 words per *Model*.

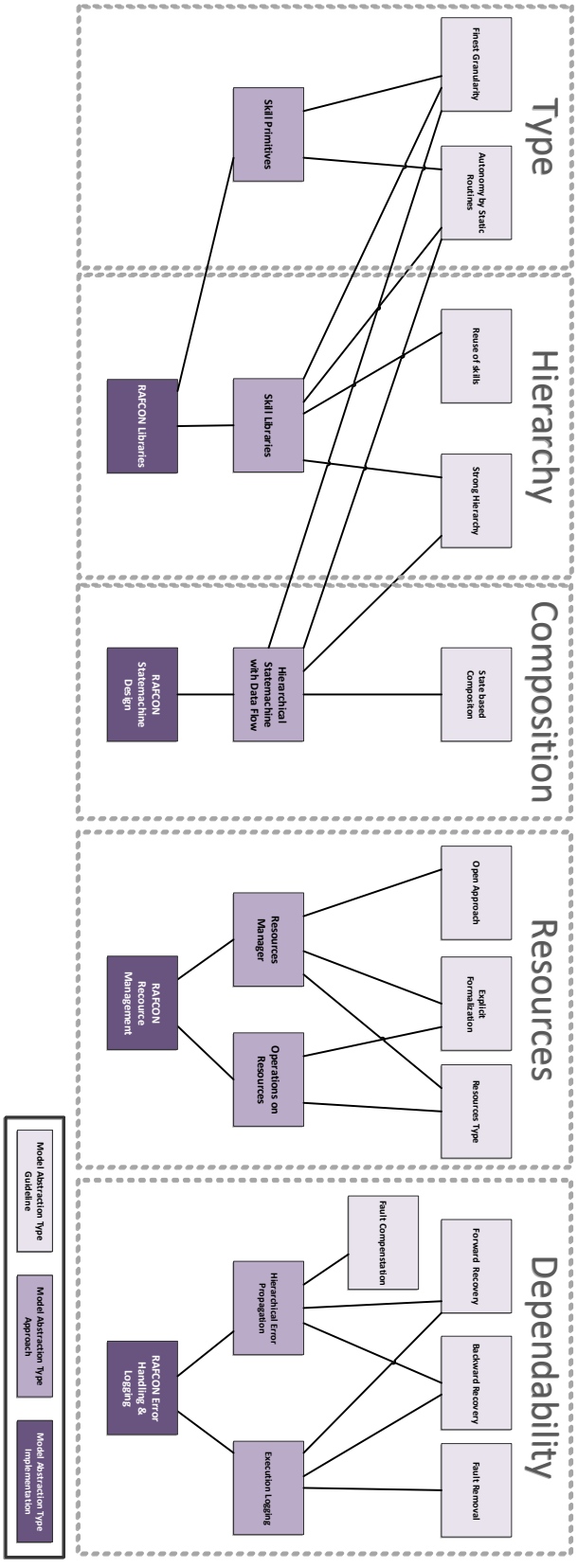

Figure 6.4: *Relations Model AIMM Skill View*

Mission View The last *View* of the *Architecture Description* is the *Mission View* from the perspective of the *Viewpoint Mission*. This is also described in full in the appendix E.

In Figure 6.5, the *Relations Model* of the *View* is displayed. As shown in Table 6.5, the *View* consists of 24 *Models*. These include 11 *Guideline Models*, 9 *Approach Models* and 4 *Implementation Models*.
The *Models* of the *Mission View* are very well separable by the *Viewpoint Aspects*. There are only very few connections between the individual *Models* of different *Aspects*. Compared to the *Views Capability* and *Skill*, the *Mission View* has a much simpler structure. Many *Models* only refer to one *Model* of the higher levels of abstraction.

	Guideline	*Approach*	*Implementation*	total
Models	11	9	4	24
Rationales	16	7	1	24
word count	2042	1324	381	3747

Table 6.5: *AIMM Mission View*

The *Mission View* contains 24 *Rationales*. The largest part of this *View* is also accounted for the *Guideline Model* with 16 *Rationales*. The *Approach Models* define 7 *Rationales* and there is only one *Rationale* in one *Implementation Model*. Overall, the *Mission View* is described with 3747 words. At 2042 words, the *Guidelines* accounts for well over half of this. The *Approach Models* is described with 1324 words and the *Implementation Models* are described very compactly with a total of 381 words.

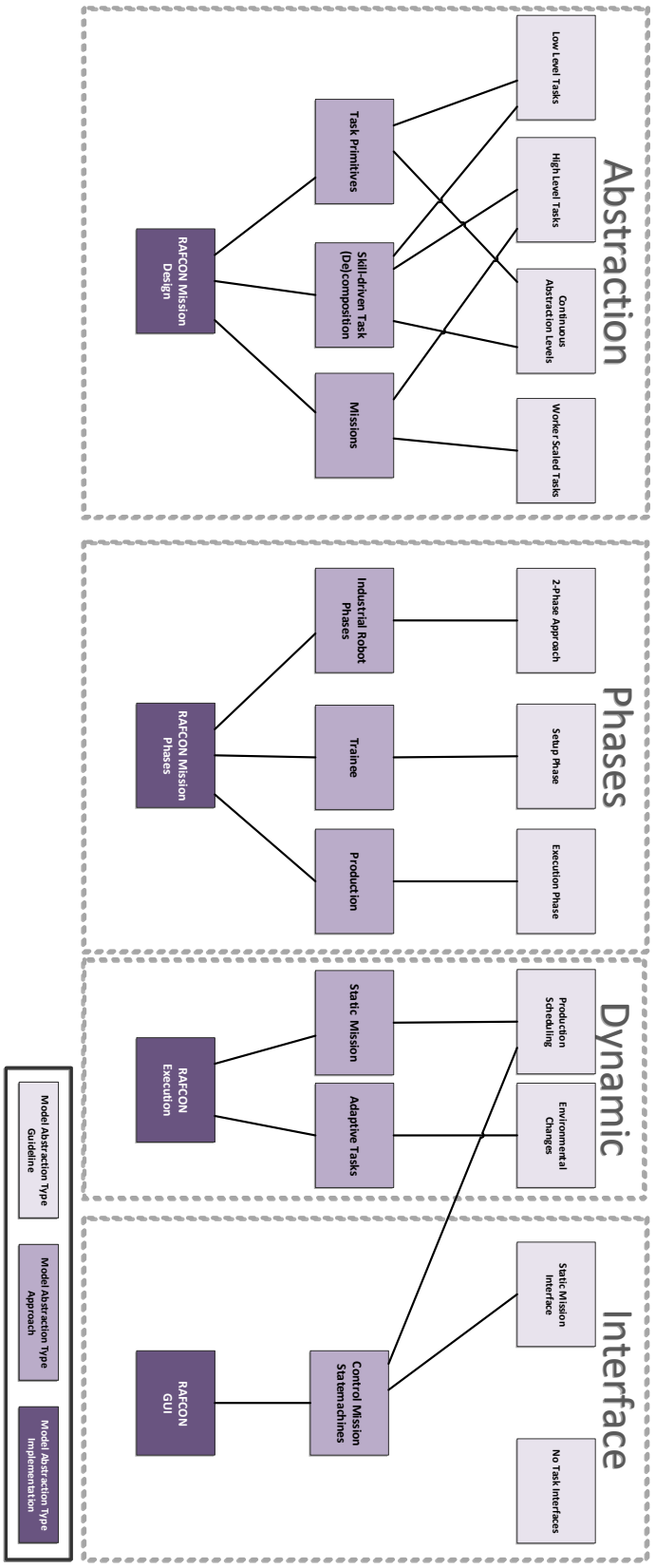

Figure 6.5: *Relations Models AIMM Mission View*

Full Architecture Description of AIMM The full *Architecture Description* of the AIMM system based on the *RoAF* is summarized in Table 6.6.

	Context	Physical	*Capability*	*Skill*	*Mission*	Total
Guideline	-	11	12	12	11	46
Approach	-	8	7	7	9	31
Implementation	-	8	4	4	4	20
Rationales	-	58	65	57	24	204
Word count	1900	6245	13931	8453	3747	34276

Table 6.6: *Overview Architecture Description AIMM*

A total of 97 *Models* were used to describe the *Architecture* of the AIMM system. Slightly less than half of the *Models* are assigned to the *Guideline* abstraction level. A total of 204 *Architecture Decisions* were documented in *Rationales*. The entire description, which contains the *Architecture Context* and the four *Views*, consists of 34276 words. According to the *Stakeholder*, the focus of the description is on the *Views Capability* and *Skill*. The most extensive *View* is the *Capability View* with 13931 words. The most compact is the *Mission View* with 3747 words.

6.1.2 Architecture Description of the Planetary Exploration Rover LRU2

The LRU2 system [112], depicted in Figure 6.6, is an autonomous *Robot* designed to develop technologies for the robotic exploration of planets.

In addition to environmental exploration, the system's *Tasks* includes the installation of infrastructure and scientific exploration. This involves using measuring instruments and collecting rock samples.

As communication with distant planets is always subject to restrictions, the aim is to achieve a very high degree of *Autonomy*. The *Robot* serves as a research platform to develop and test technologies for future space missions. The system is also used in analog missions to demonstrate the feasibility of the approaches. However, the system is not designed as a flight system. The aim of the technology development is to develop autonomous *Robots* that can work together in a team to support humans in exploring the solar system. Despite the high level of *Autonomy*, humans should be able to set the *Mission* objectives at all times and react to unforeseen situations.

Figure 6.6: *The autonomous planetary exploration rover LRU2*

This poses a particular challenge for the system design. The *Robot*'s *Autonomy* should make the technical *Complexity* manageable and, in the absence of communication, it should be able to adapt the *Mission* independently to ensure a successful execution. At the same time, it should allow humans to intervene at any time at different levels of abstraction, from *Mission* planning to remote system control. In the following, the *Mission View* of the LRU2, which reflects these challenges, is presented.

Mission View The *Architecture* of the rover LRU2 is described exclusively from the perspective of the *Mission Viewpoint*. The full description of this *View* can be found in the appendix G.

In Figure 6.7 the *Relations Model* of the *View* is shown. As displayed in Table 6.4, the *View* consists of 28 *Models*. Of these, 12 are *Guideline Models*, 10 *Approach Models* and 6 *Implementation Models*. Some of these *Models* are identical to the *Models* from the *Architecture Description* of the AIMM system. Thus, 4 identical *Models* could be identified at *Guideline* level, 3 at *Approach* level and one at *Implementation* level. A total of 8 *Models* could thus be transferred from the *Architecture Description* of the AIMM system.
The *Mission View* of the LRU2 shows, similar to the *Mission View* of the AIMM system, few links between the *Viewpoint Aspects*. However, the linking of the individual *Models* within the *Aspects* is more prominent in the LRU2 system.

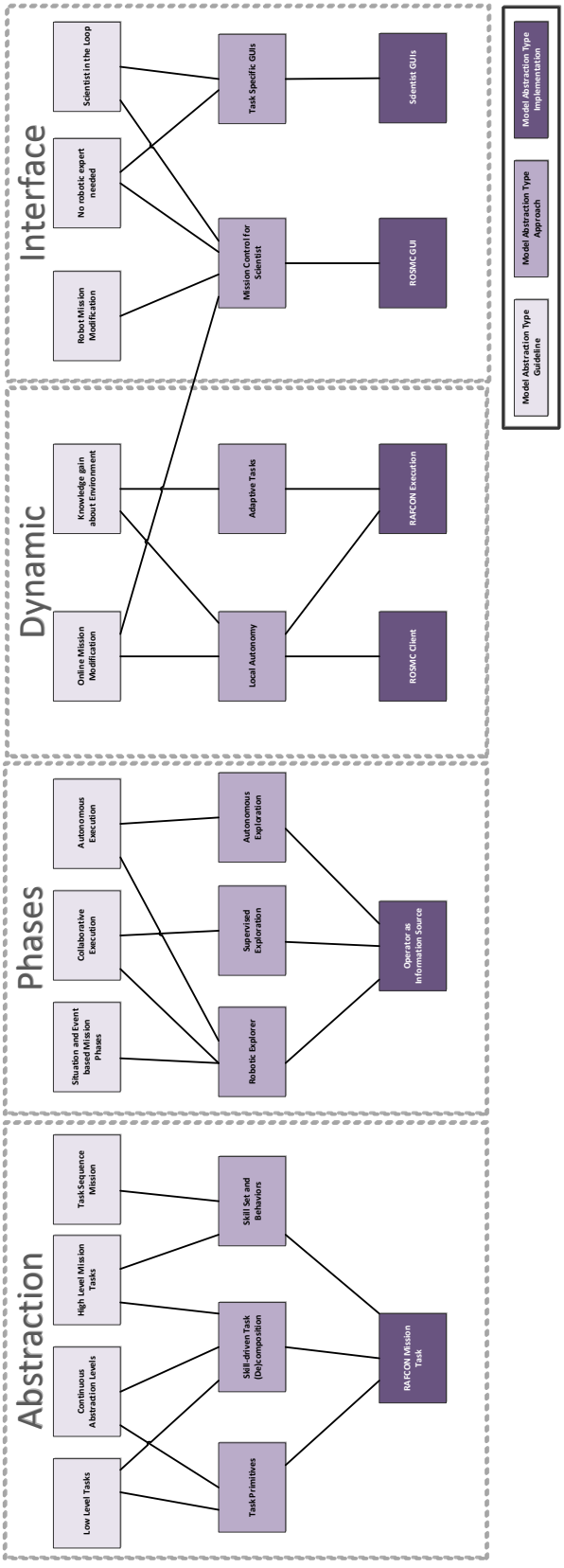

Figure 6.7: *Relations Models LRU2 Mission View*

	Guideline	*Approach*	*Implementation*	total
Models	12 (4)	10 (3)	6 (1)	28 (8)
Rationales	17 (7)	9 (2)	4 (0)	30 (9)
words	2349 (1200)	2060 (568)	1076 (75)	5485 (1843)

Table 6.7: *LRU2 Mission View*

This *Mission View* contains 30 *Rationales* of which 9 could be taken from the *Models* of the AIMM system. With 17 *Rationales*, most of the *Architecture Decisions* are justified at the *Guideline* level. At this level, with 7 adopted *Rationales*, there is also the largest overlap with the *Mission View* of the AIMM system. Among the 9 *Rationales* of the *Approach* level, there are only two identical *Rationales*. The 4 *Rationales* of the *Implementation Models* show no correspondence with the AIMM *Mission View*.
The *View* is described with a total of 5485 words, of which 1843 could be taken from the AIMM *Architecture Description*. The word count per *Model* is in the *Mission View* of the LRU2 regardless of the *Model Abstraction Type* at approx. 200 words.

Figure 6.8: *The autonomous drone ARDEA*

6.1.3 Architecture Description of the Autonomous Drone ARDEA

The ARDEA system [75], see Figure 6.8, is an autonomous multicopter that is used in the field of planetary exploration [142], but also in disaster control and humanitarian aid.

The purpose of the system is the fast, ground-independent exploration of an area. In addition, the system can be modularly equipped to collect data with various measuring instruments. The system should also be able to cooperate with other *Robots* and use them as a carrier system, for example. Due to the limited communication possibilities, but also to relieve the operator, a high degree of *Autonomy* is being sought for the ARDEA system. The size restriction of the drone poses a particular challenge. In the following, the *Physical View* of the ARDEA *Robot* is presented.

Physical View

The *Architecture* of the ARDEA drone is described exclusively by the *Viewpoint Physical*. The full description of the *View* can be found in the appendix F.

In Figure 6.9, the *Relations Model* of the *View* is shown. As displayed in Table 6.8, the *View* consists of 25 *Models*. Among these *Models* are 8 *Models* that could be taken from the *Architecture Description* of the AIMM system. 6 of these *Models* are *Guideline Models*. This means that more than half of the 11 *Guideline Models* of the ARDEA system match those of the AIMM system. Of the 9 *Approach Models*, only 2

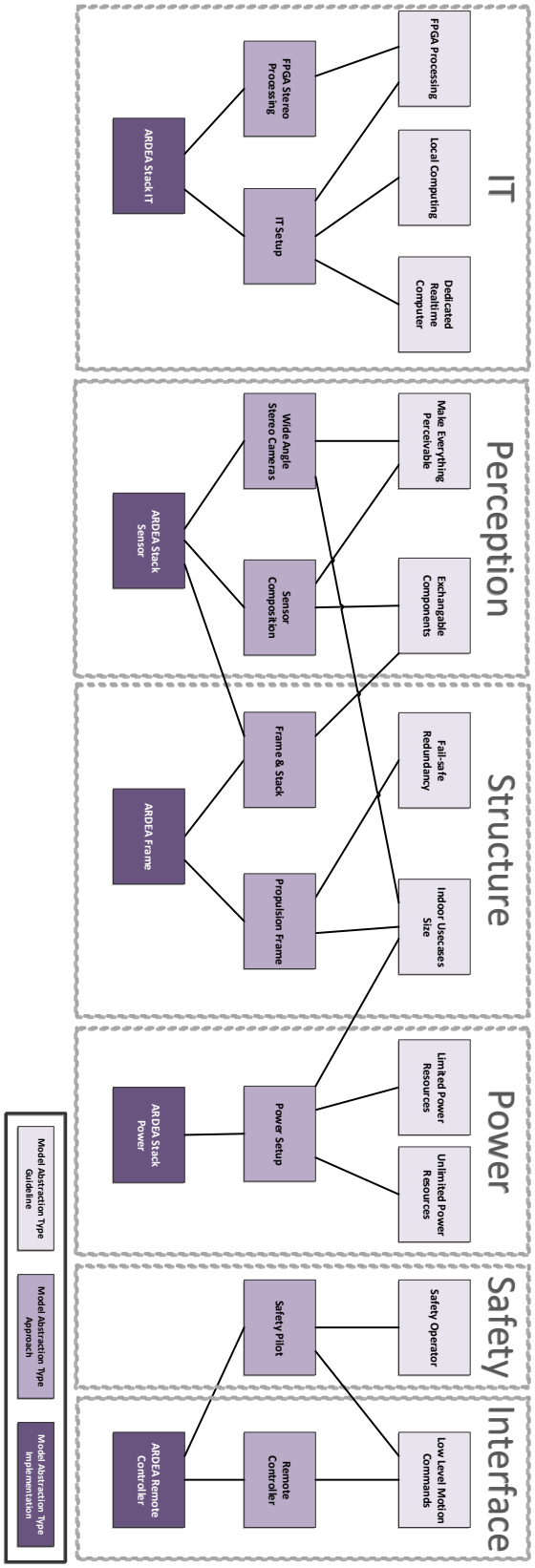

Figure 6.9: *Relations Model ARDEA Physical View*

are identical to the AIMM *Models*. The 5 *Implementation Models* have no overlap with the AIMM *Models*.
The *Models* of this *Physical View* have a stronger interconnection than those of the AIMM *Physical View*. This applies both within and between the *Viewpoint Aspects*.

	Guideline	*Approach*	*Implementation*	total
Models	11 (6)	9 (2)	5 (0)	25 (8)
Rationales	30 (18)	15 (5)	4 (0)	49 (23)
words	2228 (1519)	1701 (765)	638 (0)	4567 (2284)

Table 6.8: *ARDEA Physical View*

The *Physical View* is documented by a total of 49 *Rationales*. Of these, 23 *Rationales* were taken from the *AD* of the AIMM system. At the *Guideline* level, almost two-thirds of the *Rationales* were taken from the AIMM system, with 18 out of 30 *Rationales*. For the 15 *Rationales* of the *Approach* level, it is still one third. However, there is no overlap for the 4 *Rationales* of the *Implementation* level. The ARDEA *Physical View* has a total extent of 4567 words, of which half (2284 words) could be taken from the AIMM *Architecture Description*. In the *Guideline Models* with 1519 of 2228 words, about three quarter of the text is identical to the AIMM *Architecture Description*. In the *Implementation Models*, the proportion is significantly lower with 765 of 1701 words. Since no *Models* and *Rationales* could be adopted at the *Implementation* level, the 638 words are specific to the *Architecture Description* of the ARDEA system.

6.2 Verification

The aim of this thesis is to develop a systematic description structure for complex autonomous *Robots* that fulfills the following requirements:

1. Compactness: It must be so compact that even complex systems can be described.
2. Generality: It must enable the description of very different systems.
3. Divisibility: It must make it possible to describe the system in sub-aspects.

This section verifies that the process defined in the *RoAF* meets the above requirements. To this end, the *Architecture Descriptions* carried out are used for an exemplary verification. In a second step, a conceptual verification of the *RoAF* are carried out on the basis of the selected approaches.

6.2.1 Experimental verification

Compactness The *Architecture Description* of the AIMM system has a total of 34276 words. The various *Views* are between 3747 and 13931 words. The *Views* of the *Architecture Descriptions* of ARDEA with 4567 and LRU2 with 5485 are also in this range. The extent of a *View* is approximately that of a conference paper up to a journal publication.

Compared to the documentation of large robotic frameworks such as ARMAR-X with 4378675 words of documentation (https://armarx.humanoids.kit.edu/ date 17.1.2024) or individual software components such as RAFCON with 60918 words of documentation (RAFCON https://rafcon.readthedocs.io/ date 17.1.2024), the compactness of the *RoAF* is very clear.

This demonstrates exemplary that the *Architecture* of complex systems can be described compactly on the basis of the *RoAF*.

Generality In this thesis, very heterogeneous *Robots* from different domains and different robot categories were described. The AIMM system is a mobile manipulator that is to be used in an industrial environment. The LRU2 rover system is used for planetary exploration, which includes scientific exploration and the construction of infrastructure. The ARDEA drone is also used in planetary exploration, but is a very different type as a flight system. Based on the (partial) description of these three systems, it can thus be shown by way of example that the *RoAF* is suitable for describing *Architectures* of a wide variety of systems.

Divisibility A full *Architecture Description* was created for the AIMM system. However, the LRU2 and ARDEA systems were each only described from one *Viewpoint*. This proves by exemplary means that partial aspects of an *Architecture* can be described with the help of the *RoAF*, i.e. that the *Architecture* can be subdivided. From the *Architecture Descriptions* it is widely apparent that a further subdivision based on the *Viewpoint Aspects* is also possible. For example, the *Architectures* of a system could only be described on the basis of one *Viewpoint Aspect*.

6.2.2 Conceptual verification

Compactness The compactness of the *Architecture Description* compared to a system description lies in the abstraction, which is determined by the definition of the *Architecture* term in Definition 3.1. The system is reduced to its basic concepts and their principles of design. It is therefore not necessary to list all components, methods, interfaces, properties, etc. Instead, the architect who creates the *Architecture Description* identifies the key decisions. The *RoAF* supports this process by defining *Viewpoints*, *Concerns* and abstraction levels, but it is up to the user of the *RoAF* to decide what is included in the description and what is not. Therefore, the extent of the *AD* varies according to the personal judgment of the architect. It is therefore not possible to define a guaranteed upper limit for the extent of an *Architecture Description* according to *RoAF*. However, there is also no theoretical lower limit. It is formally permissible not to document any *Architecture Decision*. From a conceptual point of view, it is therefore possible to show that the *Architecture Description* is as compact as the architect decides.

Generality The general applicability of *RoAF* is conceptually verified by the *Robot* definition and identification as distributed software systems. In principle, the *RoAF* can be applied to systems that can be described from all four *Viewpoints*. A chatbot, for example, cannot be described from the *Viewpoint Physical*, so the full *RoAF* is not applicable to pure software. For conceptual verification, it must therefore be proven that all complex autonomous *Robots* can be described from these four *Viewpoints*.
Viewpoint Mission and *Viewpoint Physical* are derived directly from the *Robot*'s Definition 3.2 given in Chapter 3. As described in Section 3.2, this definition can in turn be understood as a superset of the common robot definitions. The definition therefore includes all *Robots* with the disadvantage that systems such as coffee machines also fall under this definition. However, it is not a problem for the *RoAF* if the process can also be applied to coffee machines. Since all *Robots* are therefore physical systems and all *Robots* fulfill *Tasks*, it is conceptually proven that the *Viewpoint Physical* and *Viewpoint Mission* can be applied to any *Robot*.
The *Viewpoints Capabilities* and *Skills* are the two software *Viewpoints* of the *RoAF*. The two *Viewpoints* are based on the *4C* [96], which were developed for distributed software systems. The conceptual verification of these two *Viewpoints* can therefore be carried out via the applicability of the *4Cs* to *Robots*. This can be demonstrated by the fact that autonomous *Robots* contain distributed software. All systems that use middleware such as ROS connect different software modules via the middleware and are therefore also distributed software systems. Since most autonomous systems use

middleware, the majority of systems can be shown to be distributed software systems. The other aspect that proves the applicability of *4Cs* to *Robots* are approaches to generatively create software for *Robots* that also use the *4Cs* as a basis. For example, the BRICS component model [26] extends the *4Cs* with a further *Concern* composition to address the various aspects of *Robot* software. This in turn implies that all systems developed with this approach can be described on the basis of the *4Cs* and thus the applicability of the *Viewpoints Capabilities* and *Skills*. It can therefore be reasonably assumed that the *RoAF* can be applied to most autonomous *Robots*.

Divisibility Subdivisibility is the central concept of the *Architecture Description* and thus also of the superordinate *Architecture Framework* according to ISO 42010 [59]. Since the *RoAF* fulfills all the requirements of an *Architecture Framework* according to ISO 42010, the general properties of an *Architecture Framework* and the *Architecture Descriptions* generated from it can be transferred to the *RoAF*. The subdivisibility of the *Architecture* is ensured by the *Viewpoint* approach. *Viewpoints* are independent by definition and each describe the overall system, taking into account the aspects defined in each case. This means that each *Architecture Description* according to ISO 42010 can be broken down into its individual *Views*, which conceptually proves the subdivisibility of the *RoAF*. The definition of common *Model Kinds*, *Viewpoint Aspects* and abstraction levels results in further structural possibilities of subdivisibility. In contrast to the *Viewpoints*, however, these are not independent of each other. For example, the different abstraction levels refer to each other. Subdivisibility via these elements is therefore only possible to a limited extent.

6.3 Validation

This section examines whether the *Architecture Description* created by the *RoAF* have their intended benefit. The following three goals were defined.

1. Comparability: It should support the comparison of different systems.
2. Evolution: It should facilitate the modification of existing systems.
3. Conception: It should make knowledge accessible for the development of new systems.

Validation is carried out in two steps, analogous to verification. First, the performed *Architecture Descriptions* are used as an example to prove that the desired goals have

been achieved. This is followed by a conceptual validation to evaluate the *RoAF* with regard to the robotic domain.

6.3.1 Experimental validation

Comparability One purpose of the *Architecture Descriptions* is to increase the comparability of systems. It should be possible to recognize and classify similarities and differences.

To this end, the AIMM and ARDEA systems are compared from the perspective of the *Viewpoint Physical*. Comparing an autonomous drone and an industrial, mobile manipulator from a hardware perspective is a major challenge, as the systems are very different. With the help of *RoAF*, the comparison can nevertheless be carried out in a structured manner by comparing the respective *Architecture Models* of one system with those of the other system. This shows that despite the heterogeneity of the systems, 8 identical *Architecture Models* can be identified. This means that with 8 out of 25 *Models* in ARDEA, around a third of the concepts are identical to those of the AIMM system. This comparatively high proportion is made possible by the abstraction levels. At the *Guideline* level, 6 out of 11 *Models* are identical, at the *Approach* level 2 out of 9 *Models* are identical and at the *Implementation* level there are no similarities. The *RoAF* therefore makes it possible to identify similar or identical concepts, although the respective implementations and their concepts no longer have any commonalities. In this way, the similarities and differences between the systems can be explicitly described.

Another aspect is that the predefined structure of the *RoAF* also describes the concepts of a system that are not the focus of research and would not be described in a system paper for reasons of space, for example. However, it is particularly important to mention these aspects in the comparison, as they represent starting points for further research or comparison with other systems. The suitability of *ADs* for system comparison is thus demonstrated by way of example.

Evolution For the development of the planetary exploration rover LRU2, the software components of the AIMM system were used to a large extent, especially for the implementation of the manipulation capabilities. Therefore, the validation that the *RoAF* can support the evolution of an existing autonomous system is demonstrated using the example of the *Mission View* from LRU2, The *Mission View* of LRU2 has diverged greatly from the *View* of the AIMM system over the years. While these two

systems had the same, identical Mission View in 2015 at the Spacebotcup [112], the one of LRU2 developed further, and now there are only 8 of 28 *Architecture Models* at the status of the ARCHES mission 2022 [99] identical. This major change is due to the extreme change in the *Mission Environment* and the resulting system adjustments. These adjustments can be seen at all abstraction levels of the *Architecture*. The explicit formulation of the *RoAF* helps to check whether *Architecture Decision* are still valid under the new conditions or need to be adapted. In the case of less serious changes to the requirements, fewer or no changes to the *Architecture* are necessary. The *RoAF* helps to identify concepts that are no longer applicable in a timely manner and replace them with suitable ones. The example of the evolution of the *Mission View* of the LRU2 shows that even large system adaptation can be carried out with the help of the *RoAF*.

Conception The third benefit of *RoAF* is support in the development of new systems. The *Architecture Description* carried out in this work contain a total of 134 *Architecture Decisions* which are discussed by 251 *Architecture Rationales*. By structuring them into *Views*, *Model Abstraction Types* and *Concerns*, the relevant *Models* for the respective decisions can still be found effectively when designing a new system. Depending on the requirements for the new system, these can serve as a template. However, they can also be identified as unsuitable for the new system based on the *Rationales*. The subdivision into different levels of abstraction also allows existing *Models* to be recombined in order to find a suitable solution for the use case. Overall, the *RoAF* thus helps to use existing knowledge for the design of new systems and to integrate new aspects into the knowledge base. Hypothetically, the *RoAF* would have considerably simplified the hardware design of the ARDEA system based on the AIMM *Architecture Description*, as one third of the concepts are directly transferable.

6.3.2 Conceptional validation

The central aim of an *Architecture Framework* is to enable the comparability and further development of existing systems, and to support the design of new systems. The *RoAF* fulfills all the requirements of an *Architecture Framework*. The approach is therefore transferable to robotics and its applicability has been demonstrated both exemplarily and conceptually in the verification process.

The fact that the approach of the *Architecture Framework* can fulfill the desired goals of comparability, further development and redesign of systems has been demonstrated

in a large number of domains. Many of the *Architecture Frameworks* refer to IT systems or software architecture. However, some examples also transfer the approach to technical systems such as the IEEE 2413 Standard for an Architectural Framework for the Internet of Things [58] or the space domain with the European Space Agency Architecture Framework [48]. It is therefore reasonable to assume that the *RoAF* will have a similar benefit in the robotics domain.

6.4 Chapter Summary

This chapter examined whether the *RoAF* and the *Architecture Descriptions* generated with it have the desired properties and offer the expected added value. To this end, a full *Architecture Description* and two partial *Architecture Descriptions* of different systems were first created based on the *RoAF*. These were presented in summarized form in this chapter. Based on these descriptions, it was then shown exemplary that the desired properties are achieved by the *RoAF*. Conceptual considerations were then used to justify that this is transferable to other systems in the domain. Then the *RoAF* was validated. Here, too, the benefit was first demonstrated using the *Architecture Descriptions* as an example. The assumption of general validity for the domain was underpinned by conceptual considerations. In the next chapter, important design decisions of the *RoAF* are discussed and further steps for future work are outlined.

Chapter seven

Discussion and Future Work

In the previous chapter it was verified that the *RoAF* has the desired properties and that the resulting *Architecture Descriptions* are useful for the comparison and development of complex robots. The used methodology, the *Architecture Framework*, allows a high grade of freedom in the design for the respective domain. Therefore, this chapter discusses important design decisions of the *RoAF*. The *Architecture Framework* was extended by *Model Abstraction Types* in order to describe the *Robot Architecture* through different levels of abstraction. When choosing the *Viewpoint* care was taken to keep them independent. Thus, the *RoAF* does not use *Correspondence* rules between individual *Views*. In order to achieve general applicability, the *Model Kinds* of the *RoAF* were kept generic. This enables any *Architecture* decisions to be documented. Another design decision in the *RoAF* is to also capture simple *AD Elements*. The goal is always to create a complete *Architecture Description* and to also document decisions that were only made for practical reasons.
A side effect of the *RoAF* is that by documenting the *Architecture Decisions*, but also through the elements of the framework, relevant aspects of *Robots* are identified and described. Not all of these aspects are well represented in current research. Thus, the *RoAF* helps to identify relevant topics for future research. The chapter concludes with an outlook on future work, in particular how the *RoAF* can contribute to the systematic development of autonomous *Robots* in the long term.

7.1 Separation of Guideline, Approach and Implementation

The structuring into the *Model Abstraction Types Guideline*, *Approach* and *Implementation* is elementary for the benefit of the *Architecture Descriptions*. At the *Guideline* level, desired properties and procedures can still be defined relatively independently. Similarities but also differences between systems can be easily identified at this level. At the *Approach* level, on the other hand, it can be seen that a compromise often has to be found between different conceptual goals in robotics. Many *Models* at the *Approach* level therefore do not describe the *Approach* of how a specific *Guideline* can be achieved. Rather, they describe how the various requirements for the specific system are linked together. Different *Guidelines* are therefore often addressed for one *Approach*. Despite the same *Guidelines*, systems can differ greatly in their *Architecture* at the *Approach* level. It can also be observed at the transition to the *Implementation* level that different *Approaches* with a *Implementation Model* are often connected. This shows that at the *Implementation* level, a single component is not used for each *Approach*, but the software can often implement several *Approaches*. This reduces complexity. Conversely, however, it is also apparent that *Approach* are more likely to be selected for systems that can be implemented with the existing implementation solution. The complexity of the *Architecture* of *Robots* becomes clear due to the large number of links contained in the presented *Architecture Descriptions*. This explicitly shows that many architectural decisions, whether at the level of the *Approach* or at the level of *Implementation*, take into account a variety of aspects, even across *Aspects*. A large part of the *Architecture* in robotics is to address different, sometimes conflicting requirements. Even with identical *Guidelines*, a wide variety of *Architectures* can emerge. It indicates that *Architecture* in robotics is probably individual for a *Robot* or at least for a narrowly defined subset of systems. It is therefore unlikely that a universally valid reference *Architecture* can be found.

7.2 Independence between the Views

The *Views* achieve a high degree of separation. This allows concepts to be described and compared independently of their global dependencies. The division also correlates with the *Stakeholders* of a *Robot*. This means that different *Developer* can each find a full *Architecture Description* of the system from their perspective in a *View*. The *RoAF* restricts the relationships between the *Viewpoints* to a few generally valid

dependencies. Individual relationships across *View* boundaries cannot be captured with the *RoAF*. This simplifies the *Architecture Description*, improves comparability and makes it possible to create individual *Views* of an *Architecture*.
However, this does not mean that dependencies do not exist and that *Views* can be developed completely independently of each other. For example, a change in the *Capability View* can result in a significantly higher computing power requirement, which would have to be taken into account in the *Physical View*.

This relationship is not modeled in the *RoAF* and is therefore not directly apparent from the *Architecture Description*. Since the *RoAF* is used to describe the *Architecture* of existing systems it is guaranteed that the *Views* fit together and all dependencies are taken into account. However, if a system is only further developed on the basis of the *Architecture Description*, it can happen that there are effects on other *Views* without this being apparent from the *Architecture Description*.

7.3 Generic Model Kinds

Model Kinds of the *RoAF* are very generic. Essentially, a *Model Kind* is defined by the *Viewpoint Aspect* and the *Model Abstraction Type*. There are not much further formal restrictions. This means that *Architecture Decisions* can simply be formulated in free text. In addition, *Architecture Rationales* are available to justify the *Architecture Decision*. This very free form has the advantage that any decisions can be documented. Stronger formalization requires structures, which increases the risk that not all decisions can be documented, or that *Model Kinds* cannot be applied to all systems. However, due to the very free form of representation, there is a risk that similar solutions will not be recognized as such because they are described in a different way. This is particularly relevant when a large number of *Architecture Descriptions* are available. But these descriptions can then be used to identify suitable structures and extend the *RoAF* accordingly. Since this information is currently difficult to access, and the problem is manageable with a small number of *Architecture Descriptions*, this free form of *Models* was chosen.

7.4 Simplicity of Architecture

The *RoAF* enables the systematic capture and description of the *Architecture* of a system. However, this does not ensure that the documented *Architecture* is also of high quality and, for example, that it stands up to scientific review. On the contrary, the fact that the *RoAF* ensures very broad coverage of a wide variety of aspects when using all *Viewpoints* and *Concerns* means that aspects that are not the focus of the respective system are also addressed in every current research system. In addition to innovative, new solutions, a number of simple concepts can also be identified that were used to solve certain problems with minimal effort.
For the *RoAF*, a conscious decision was made to also document these *Architecture Decisions* and even to promote them through the strong networking in the *Viewpoints* in order to create descriptions that are as complete as possible. There are two main reasons for this. First, it is often very valuable for the community and each individual *Robot* architect with limited resources to find simple solutions for topics that are not within the main expertise of the respective team. For a functioning system, all aspects must be solved, but there is no documented basic solution that can be easily implemented; instead, each new team solves all problems again each time with as little effort as possible before they can concentrate on their actual work. These simple solutions are then usually not published. The aim of *RoAF* is therefore also to document these simple solutions in order to make this knowledge accessible. Systematic evaluation is also often difficult for the publication of system solutions, as no basic solution exists as a reference.

The second aspect why complete *Architecture Descriptions* are important is that if the focus is on individual sub-areas, composability is not guaranteed. For example, an *Architecture Decisions* in one area can prevent solutions in other areas. To advance the development of autonomous systems as a whole, it is therefore important to check compatibility with the overall architecture. Only through a full description of the *Architecture* can it be recognized when progress in one area leads to problems in other areas.

7.5 Identification of Research Topics

The *RoAF* is a structured analysis of *Robots*. A central goal was to identify aspects that are applicable to as many *Robots* as possible. Indirectly, however, this also means

that these aspects play a major role in robotics, as every system requires a solution. Nevertheless, it has been found that not all aspects are established research topics. For example, the *Viewpoint Aspect* World Model of *Viewpoint Capabilities* is essential for any autonomous *Robot*. Nevertheless, there has been little information in the literature about what a world model is and what approaches exist. Based on the *RoAF*, this topic was identified as underrepresented and was established as one of the three research topics of the Transferable Explainable Knowledge group of the DLR Institute of Robotics and Mechatronics. As a result, two doctoral positions were created and the first publications have already been published [101], [100]. The *RoAF* thus also makes it possible to identify important topics in robotics that are currently still little covered in the literature, thereby helping to close important knowledge gaps.

7.6 Future Work

In this thesis, a process was developed that allows to describe the *Architecture* of complex *Robots* systematically. This makes previously hidden knowledge about robot design accessible and makes it possible to learn from existing solutions. The current situation, where the development of a *Robot* is based on the knowledge and experience of the individual system engineer, will initially only change insofar as it becomes easier to gain knowledge about autonomous *Robots*. However, this will not lead directly to the systematic development of autonomous *Robots*.

Several steps are necessary to achieve this goal. These are shown schematically in Figure 7.1.

The first step has been taken with *RoAF*. It is now possible to describe the *Architecture* of autonomous systems. However, the process can only be used by people who know the *Architecture* of the system.

The next step is therefore for other system architects to describe their systems using the *RoAF*. Based on these further descriptions, adjustments to the *RoAF* will probably be necessary, as the *RoAF* is only based on the available information and the collected knowledge about the analyzed systems. This can be done by extending the *Viewpoints* or the *Aspects* under consideration. More formalized *Model Kinds* could help to identify and reuse existing concepts. This iterative process will gradually improve the *RoAF* so that all relevant aspects of systems can be considered.

In the next step, patterns could be identified on the basis of the *Architecture Descriptions* and their comparability. It is therefore reasonable that certain suitable

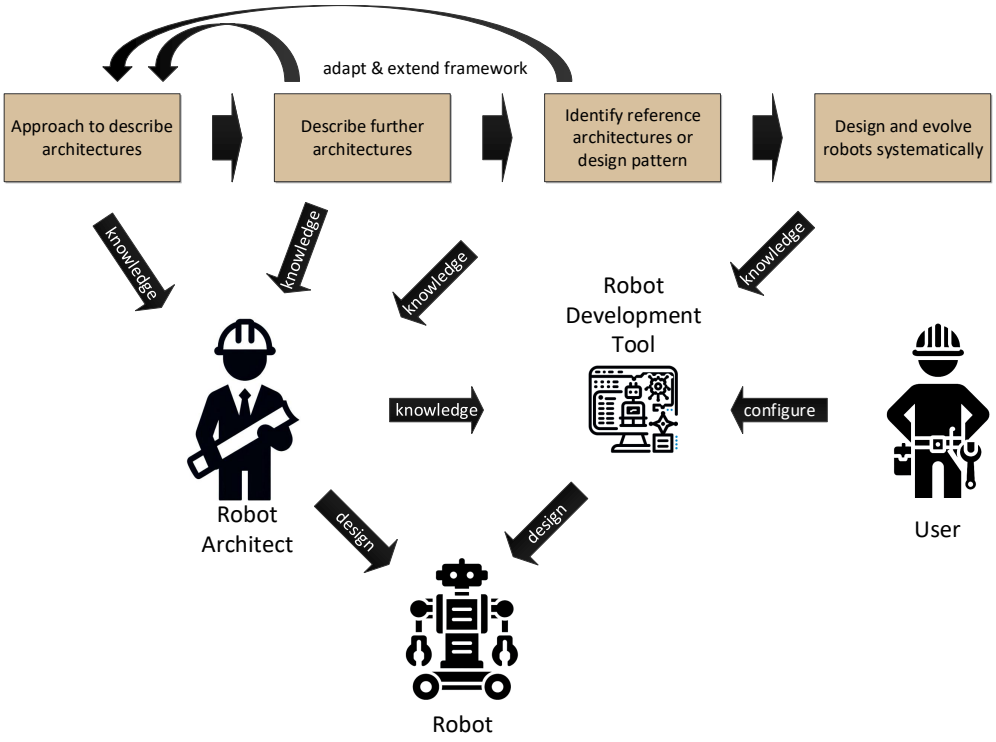

Figure 7.1: *Steps towards a systematic approach to the design of autonomous systems: The first step of developing an approach to describe architectures of autonomous systems was presented in this thesis. The next step is to describe further architectures and to adapt and extend the framework accordingly. Based on this description architecture design patterns can be identified in a next step. These three steps make knowledge accessible to the Robot Architect, the design of the robot itself is not addressed. To achieve a systematic robot design, a further step is needed, where tools are developed that allow the user to design a robot by configuration.*

Architecture concepts can be identified through similar system requirements, environments or *Tasks*. These *Architecture* patterns should then in turn be integrated into the *RoAF* by extending the *RoAF* via metamodels. This also simplifies the description of individual *Architectures*, as complete patterns can be identified for parts of the *Architecture* and not every aspect has to be described from scratch.

All these steps will help to improve the *RoAF* and make more knowledge accessible. However, it remains the case that existing knowledge is extracted from existing systems and made accessible to the system architect. The active part, i.e. the design of new systems or the further development of existing systems, remains limited to the skills of the individual developer.

To allow for interpolation into new systems, i.e. to take the fourth step, the *RoAF* alone is not sufficient. It is necessary to represent not only the result, but also the

development process itself. In addition, simplifications through abstraction and separation of the concerns are no longer permissible and all existing dependencies must be taken into account. If the *RoAF* were extended accordingly, the resulting complexity would probably lead to important objectives such as comparability no longer being met.

It is therefore likely that other approaches will be required for the design of systems. As described in Chapter 2, there are already approaches in the form of model-based software frameworks that can at least be used to systematically create the system's software. These approaches are not yet able to cover all the requirements of autonomous systems. However, the *RoAF* could help to identify whether the requirements for the system can be met by the framework. This could be done, for example, using architecture patterns that are already used in model-based system development [102]. It remains to be seen whether these approaches are also suitable for other aspects of a system, e.g. the sensor concept. Therefore, a combination of different approaches will probably be necessary to create and further develop autonomous systems without expert knowledge. In the long term, the *RoAF* could therefore serve as the basis for a meta-level for composing different frameworks.

Appendix A

Architecture Context of the Autonomous Industrial Mobile Manipulator AIMM

This chapter describes the *Architecture Context* of the autonomous, mobile manipulator AIMM based on the developed *RoAF* and identifies the system-of-interest, the system environment, the *Stakeholders* and relevant system-specific *Concerns*.

A.1 System-of-Interest

The AIMM *Robot* is a research platform for investigating mobile manipulation in an industrial context. The *Robot* was developed as part of the EU project TAPAS in 2011. The task of the *Robot* in the project was to solve logistics *Tasks* and simple assembly *Tasks* in a pump factory. The idea was to take on *Tasks* that had not previously been automated due to the small quantities involved. The *Robot* was therefore not supposed to perform just one *Task*, but various *Tasks* at different workstations. The biggest challenge was therefore to make the system so flexible that it could carry out different *Tasks* in a new environment in a short period of time. At the end of the project, this was successfully demonstrated on a real production site [12].

The lesson learned from this project was that it is possible to develop a *Robot* that

Figure A.1: *The autonomous industrial mobile manipulator AIMM*

can flexibly solve *Tasks*. But the flexibility and dependability of the system could only be achieved by many experts on site, who adapted various modules and behaviors to the *Tasks* at hand. Since then, the aim of the AIMM research platform has been to research autonomy approaches that make it possible to operate the system without *Robot* experts. The main system's publication is Dömel et al. [35].

Usecase and Environment - System Boundaries to Physical Environment and Mission Environment The *Physical Environment* of a *Robot* in an industrial setting, such as a factory floor, has domain-specific characteristics that are important for the *Architecture* of a *Robot*. Most industrial sites are static to a certain degree. This means that large machines or fully automated cells are set up once and are only slightly changed until the end of their operating time. The general structure of a factory is also static. The warehouse will not change its location overnight. On the other hand, a production facility can be very dynamic, for example with fast-moving forklift trucks. However, as this is also dangerous for human workers, they are usually restricted to certain areas. Large parts of a production facility are not dynamic, but still change over time. For example, pallets of parts are placed in the environment, or workers place boxes or tools on tables. These quasi-static environments, with a combination of truly static things like tables and machines and non-dynamic but changing objects like boxes, pallets and tools, are the most common environments in which people spend time in a production facility. For human workers, it does not matter where

exactly a box is located, as flexibility is not a problem for them. For *Robots*, however, this poses a real challenge. In addition to these domain-specific characteristics, a *Robot* in a factory has to deal with a real environment. In contrast to laboratory conditions, problems such as changing light conditions, battery management and connection losses to external networks must be taken into account. To summarize, an industrial environment is a partially known, semi-structured environment that is usually quasi-static.

The *Tasks* in industrial applications are well documented. There are usually very precise work descriptions for human workers. The *Mission Environment* therefore already provides a certain degree of segmentation into *Subtasks*. The challenge for the *Robot* is to fulfill these *Subtasks* in the *Physical Environment* and provide an interface to execute new sequences of these *Subtasks*.

A.2 Stakeholders

The AIMM *Robot* is purely a research platform. AIMM acts as a demonstrator on which new approaches can be developed and tested. The aim is therefore not to develop a prototype that will subsequently be marketed. Instead, the focus is on scientists who want to use the system to research new approaches. The scientists take on the role of both *Developer* and *User*.

Nevertheless, the *Concerns* of end users are also important in the development of the research platform. They form the framework for the long-term development goal. *Stakeholders* who have an interest in these goals are therefore also identified, even if they are not directly involved. These future *Stakeholders* essentially correspond to the *Stakeholders* from the *User* group identified in the *RoAF*, transferred to the industrial context.

Future Stakeholder the end users:

Operator are persons who operate systems such as AIMM. Operators are not expected to have any special knowledge of robotics. As a rule, they are workers who also perform other tasks in the factory.

Coworker are people who work together with the system. This allows work steps to be shared between *Robots* and humans. One example of this is the supply of assembly workstations by the *Robot* with the required parts and tools. It is also conceivable that the coworker could carry out *Tasks* collaboratively with the

Robot. In this case, the *Robot* can act as a third hand or take on simpler *Tasks* in a joint assembly process.

Maintainer are people who are responsible for carrying out maintenance work on the systems. This includes, for example, changing batteries, calibrating sensors and replacing wearing parts.

System Integrator are people who are responsible for integrating a system into an industrial plant. The main task of the system integrator is to connect the system with the factory's control system. For complex *Tasks*, the system integrator must also integrate process-related knowledge into the *Robot*.

Factory Owner are the people who buy the system. They are interested in the cost-efficient execution of *Tasks* by the *Robot*.

The various future *Stakeholders* provide AIMM with the rough direction in which the development of the system should be driven forward. The individual points are weighted differently. The current state of the art is concerned with feasibility in principle; economic viability, for example, is not yet a factor. However, due to limited resources, it is not possible to work on all of the required technologies; instead, AIMM concentrates on certain sub-areas. The roles of the *RoAF* are not divided among the people, but each person must take on a certain proportion of each role, from module developer to operator.

The person-specific separation takes place at a specialist level, so that the AIMM *Stakeholders* can be identified as follows:

Stakeholder AIMM:

Perception Researcher are persons who deal with methods and approaches in the context of the perception of the *Physical Environment*. The two main focuses of AIMM are modeling the partially unknown environment on the one hand and semantic scene analysis on the other.

Manipulation Researcher are people who deal with methods and approaches for manipulating the *Physical Environment*. The AIMM system focuses on efficient movement planning and the development of robust manipulation strategies.

Autonomy Researcher are persons dealing with methods and approaches for the autonomy of the *Robot*. The focus of the AIMM is on the flexibility and robustness of *Tasks* execution, even with incomplete or incorrect knowledge of the environment or the system.

System Architecture Researcher are people who deal with methods and approaches to the *Architecture* of *Robots*. The focus of the AIMM is on identifying and developing general concepts for the *Architecture* of *Robots*.

The topics to be investigated on the AIMM research platform are defined by the relevant *Stakeholders*. In order to meet the requirements of future *Stakeholders*, further technologies must be investigated in addition to the core topics. The following list therefore refers to the necessary topics, which are not considered from a research perspective.

No Stakeholders of AIMM:

Hardware Researcher are people who deal with methods and approaches for developing new hardware components. Purchased components are used at AIMM.

Control Researcher are people who deal with methods and approaches for the development of new control technology concepts. At AIMM, the controllers that are included in the purchased components are used.

Safety Researcher are persons who deal with the development of new safety concepts. The AIMM system is used exclusively in monitored operation, whereby safety is ensured by a person with an emergency stop.

HRI Researcher are persons who deal with new concepts for human-robot interaction. At AIMM, only absolute experts operate the system. Interaction with the *Robot* takes place on a very technical level.

Fleet Management/Factory Control Researcher The activity of "Fleet Management/Factory Control Researcher" involves the integration of autonomous *Robots* into a higher-level system, such as a factory. In the AIMM system, there is no higher-level instance.

A.3 Concerns

Similar to the *Stakeholders*, two areas can also be distinguished for the *Concerns*: On the one hand, there is the area of future requirements, which guides development, and on the other, the area of specific requirements of current *Stakeholders*. The RoAF *Concerns* naturally also exist in the AIMM. However, the *Stakeholders* of the system result in a weighting of the *Concerns*.

A.3.1 Future Stakeholders' Concerns

This section describes the *Concerns* of future *Stakeholders*, categorized according to the most important *User Concerns*. These form the scope for the development of the AIMM system.

Flexibility The aim of the AIMM system is to solve a variety of different logistics and assembly *Tasks* autonomously. The system must be able to cope with all expected environmental conditions in a factory.

Dependability AIMM must perform its *Tasks* reliably. To achieve this, the system must be so robust that any regularly occurring problems, such as poor lighting conditions, environmental changes, etc., do not jeopardize the successful completion of the *Tasks*. In the event of unforeseeable and unrecoverable faults, such as a power failure in the system, the system must behave in such a way that no damage occurs.

Usability The operation of AIMM should be intuitive and simple. This applies both to controlling the *Robot* and to working with the system. It must also be possible to hand over new *Tasks* to the system without expert knowledge and the system must be able to communicate its status to its environment in an understandable way.

A.3.2 Concerns of the Scientist

At AIMM, the scientists are both developers and users. The long-term goal is to fulfill the *Concerns* of future users. However, the development and the work of the researchers also result in direct *Concerns* for the system. As the researchers also take on the role of *Developer*, the researchers' *Concerns* are assigned to the five main *Robotic Concerns* below.

Flexibility AIMM must be flexible in several respects. In terms of system boundaries, the system must be able to solve different use cases in different environments in order to validate and demonstrate the scientific approaches. But AIMM must also be flexible within the system boundaries. Newly developed components often have to be integrated or other components replaced. From a scientific point of view, it is also necessary to compare different system configurations with each other.

Dependability AIMM must have a certain degree of dependability in order to be able to work with the system. The requirements are not as high as in real industrial use, but with an interaction of several hundred components, either the individual components must be very reliable or strategies must be in place to compensate for the failure of individual components. Otherwise it is not possible to operate the system properly.

Usability As AIMM is only used by robotics experts, the requirements for the system differ from those used by end users. Operation must be less intuitive and can be based on knowledge. Nevertheless, it is also important for the expert to be able to monitor and control the system status efficiently.

Complexity Even if experts can handle a significantly higher level of complexity than inexperienced operators, sooner or later a limit is reached that can no longer be controlled. One of the scientists' *Concerns* is therefore to keep complexity as low as possible.

Autonomy Autonomy is one of the most important research goals of the AIMM system. Therefore, methods and approaches are being researched to increase the degree of autonomy. In addition, the autonomy of AIMM is an important instrument for addressing the other *Concerns*.

Appendix B

AIMM Physical View

This chapter describes the *Physical View* of the AIMM system. The AIMM system has no *Stakeholders* with a research interest in this perspective of the system. The concepts presented are therefore practical solutions for a functional research platform with a focus on perception, manipulation and autonomy. First, an overview of all identified *Models* and their *Relations* to each other is given. The individual *Architecture Models* are then presented.

B.1 AIMM Physical Overview

***Guideline Models*:**

- Keep It Simple
- Use Off-the-Shelf Components
- Exchangable Components
- Make Everything Perceivable
- Sensor Groups
- Local Computing
- Unlimited Computational Resources
- Unlimited Power Resources
- Safety Operator
- Low Level Motion Commands
- Simple Status Display

***Approach Models*:**

- Mobile Manipulator
- Head, Hand & Navigation
- Processing
- Network
- Power Distribution
- Safety
- LED Status Display
- Remote Controller

***Implementation Models*:**

- KUKA omniRob Structure
- KUKA omniRob Power
- KUKA omniRob Safety
- KUKA omniRob Remote
- Amtec Pan-Tilt Unit
- Roboception rc_visard 65
- LED-Projektor
- COM Express Boards

B.2 AIMM Physical Structure

B.2.1 Keep It Simple

Model Name: Keep It Simple	Model Kind: VP-M1G	Model Type: Guideline	ID: G1
Addressed Aspects: VP-A1			
Description: The concept is to keep the system kinematically as simple as possible. This reduces the mechatronic complexity of the system. Nevertheless, the system requirements must be met.			

Rationale 1: ***Simple kinematics reduces the system complexity***
A complex kinematic structure significantly increases the system complexity. The reason

Figure B.1: *The Relations of the AIMM Physical View Models*

for this is the direct influence on the software components. Complex kinematics generally require more complex controllers, schedulers, etc. The reduction in mechanical complexity leads to a significant reduction in overall complexity.

Rationale 2: ***No Humanoid***
One difficulty with this approach is finding the simplest solution that meets the requirements. One approach is therefore to take human kinematics as a basis. This implicitly fulfills the requirements, as the relevant environments have been designed for human kinematics. However, this greatly increases the complexity of the system, which is why this approach is not pursued.

Rationale 3: ***No scientific interest in mechatronics***
The system does not have a *Stakeholder* who is involved in the mechatronic design of the system. This approach minimizes the effort in this area.

Rationale 4: ***Soft system requirements***
The system is a research platform. This means that both the environment and the *Tasks* can be adapted to the system's capabilities to a certain extent. The requirements for the system are therefore less critical than for a system for productive use in a factory.

B.2.2 Use Off-the-Shelf Components

Model Name:	Model Kind:	Model Type:	ID:
Use Off-the-Shelf Components	VP-M1G	Guideline	G2
Addressed Aspects: VP-A1, VP-A2, VP-A3, VP-A4, VP-A5, VP-A6			
Description: The hardware of the *Robot* should consist of commercially available components. Custom-made products should be avoided wherever possible.			

Rationale 5: ***Suboptimal hardware components***
One disadvantage that must be taken into account with this concept is that the hardware is not optimized for the exact application. There are usually significant disadvantages compared to a custom-made product. For example, the required installation space is larger, the components do not fit together without adapters and there are limitations in terms of performance.

Rationale 6: ***Faster & cheaper development***
Off-the-shelf components are much more cost-effective than custom-made products due to series production. They are also available more quickly and are easier to maintain.

Rationale 7: ***Increased dependability***
Finished products are extensively tested and any weak points are eliminated. This makes the products more reliable than custom-made products. As the hardware forms the basis of the system and failures cannot usually be compensated for, dependability is particularly

important here.

Rationale 8: ***Interesting for future stakeholder***
Even if the *Robot* is purely a research platform, it should also be of interest to future *Stakeholders* if possible. If the *Robot*'s hardware is already commercially available, this is a strong argument for the feasibility of the concept.

B.2.3 Mobile Manipulator

Model Name:	Model Kind:	Model Type:	ID:
Mobile Manipulator	VP-M1A	Approach	A1
Addressed Aspects: VP-A1			
Description: The kinematic structure is a mobile manipulator. A mobile manipulator consists of a manipulator and a mobile platform. Based on the requirements of the use case and the concepts applied, a holonomic platform was selected. This can move in flat environments with the help of Mecanum wheels. A torque-controlled arm with a redundant axis was selected as the manipulator. A parallel gripper with two fingers is used as the gripper.			
Applies: G1, G2			

Rationale 9: ***Selection of a the mobile platform with mechanum wheels***
Due to the requirements to be fulfilled by the system, the mobility of the *Robot* is a fundamental prerequisite. The simplest way to make a *Robot* mobile is to use wheel-based platforms. The kinematics of the platform is crucial here. Since uncertainties are to be expected, corrective movements must be easily possible. The minimum requirement is therefore omnidirectional kinematics. Platforms based on Mechanum wheels meet this requirement. At the same time, the mechanical design is very simple. In addition, this technology is tried and tested and is sold commercially by various suppliers. The disadvantage of this concept is that the surface must be relatively level. However, this can be considered as given for this application.

Rationale 10: ***Single manipulator***
The kinematic configuration provides only one manipulator. This reduces the complexity of the system compared to systems with multiple arms. However, this also reduces the system's capabilities. In particular, handling larger objects and assembly operations without fixtures are difficult or impossible.

Rationale 11: ***Selection of the manipulator***
The manipulator interacts with its environment. Since unknown obstacles and inaccuracies are to be expected, the manipulator must be force-controlled in order to detect collisions and

compensate for inaccuracies. A redundant degree of freedom makes it possible to approach a point in the workspace in different manipulator configurations, which significantly increases the flexibility of the system with only a slight increase in complexity. Manipulators with these features are commercially available. A manipulator on a mobile platform has a significantly smaller workspace than a human, for example. Additional joints or linear axes would be required to approach this working space. This significantly increases the complexity of the system. These limitations are therefore accepted for the research platform.

Rationale 12: ***Selection of the gripper***
A 2-finger parallel gripper was chosen as the gripper. This is one of the simplest forms for grasping various objects. Compared to a multi-finger gripper or a hand, the complexity is greatly reduced. These grippers have been used in industry for a long time and are therefore commercially available and very reliable.

Rationale 13: ***No suction gripper***
A suction pad is also a simple and very flexible solution for grasping objects. The principle works particularly robustly on large surfaces. The disadvantage is that the exact grasp position cannot be determined. This makes it difficult to achieve the precision required for assembly tasks.

B.2.4 KUKA omniRob Structure

Model Name:	Model Kind:	Model Type:	ID:
KUKA omniRob Structure	VP-M1I	Implementation	I1
Addressed Aspects: VP-A1, VP-A2			
Description: The KUKA omniRob was chosen as the basis for the AIMM system. The KUKA omniRob therefore implements the physical structure of the AIMM system. The KUKA omniRob is equipped with Mechanum wheels, which fulfill the requirements of a holonomic platform. The mounted KUKA LBR4+ is a force-controlled manipulator that also fulfills the requirements of the AIMM concept. Missing sensors and computing infrastructure can be added later, as both installation space and energy are available.			
Implements: A1, A2			

B.3 AIMM Physical Perception

B.3.1 Make Everything Perceivable

Model Name:	Model Kind:	Model Type:	ID:
Make Everything Perceivable	VP-M2G	Guideline	G3
Addressed Aspects: VP-A2			
Description: The system's ability to perceive is an important part of the interface between the *Robot* and its physical environment. Perception is necessary to recognize uncertainties or faulty information and to monitor processes. The more information available to the system, the better the *Robot* can react. The concept is therefore to make as much information as technically possible perceptible. This means that the system must be equipped with many sensors.			

Rationale 14: ***Additional information does no harm***
In contrast to mechatronic components, additional sensors do not lead to a significant increase in complexity. Sensors offer the possibility of using additional information, but they do not force this. It therefore makes sense to generate as much sensor information as possible.

Rationale 15: ***No perfect sensor available - combine them***
There is currently no sensor that delivers good results under all realistic conditions. Active sensors, for example, are sensitive to light, stereo sensors require texture, ultrasonic sensors are material-dependent, etc. The conclusion is that there is no perfect sensor, but that each concept has its strengths and weaknesses. Ideally, different sensor types should therefore be used in order to obtain good data in every situation.

Rationale 16: ***Configuration of the sensors***
Sensors can be adapted to a task by configuring the parameters (base distance) and using accessories (lenses). As this adaptation is static, it may be necessary to use the same sensors in different configurations.

Rationale 17: ***Useful redundancy***
Even if sensors provide identical information, i.e. are redundant for the perception of the environment, they can still be useful for the system. For example, the calibration of the sensors can be checked. Different viewing angles of the same object can also be used to improve accuracy.

Rationale 18: ***Praktical limits***
There are practical limits to the number of sensors. For example, there is not an unlimited number of assembly positions. At the end effector in particular, sensors considerably restrict the *Robot*'s workspace. Another limitation is data transmission and processing. Cameras in particular generate large amounts of data that can quickly overwhelm the system's technical infrastructure.

B.3.2 Exchangeable Components

Model Name:	Model Kind:	Model Type:	ID:
Exchangeable Components	VP-M2G	Guideline	G4
Addressed Aspects: VP-A1, VP-A2			
Description: The mechanical interchangeability of the sensors is an important concept for AIMM. As the perfect configuration for mobile systems has not yet been found, it must be possible to adapt the system's sensor configuration without great effort.			

Rationale 19: ***Part of the research***
The development of suitable sensor technology for a mobile manipulator is the subject of current research. The combination of different sensor types in particular offers interesting questions. In order to carry out these experiments, it must be possible to convert the system without great effort.

Rationale 20: ***New sensors***
The development of sensors is also progressing rapidly. New commercial sensors that can provide better data are regularly available. The concept considerably simplifies the integration of these new sensors.

Rationale 21: ***Lower integration density***
The flexible hardware structure requires installation space. Compared to a fixed solution, the integration density is reduced. Depending on the position of the sensors, this disadvantage is more or less relevant. Especially at the end effector of the robot, the size has a strong influence on the workspace.

Rationale 22: ***More complex calibration***
In order to use the sensor data, the position of each sensor must be known exactly. The sensors are therefore calibrated in relation to each other and to the *Robot*. This process is very time-consuming if many different sensors are used at different mounting positions.

B.3.3 Sensor Groups

Model Name:	Model Kind:	Model Type:	ID:
Sensor Groups	VP-M2G	Guideline	G5
Addressed Aspects: VP-A1, VP-A2			
Description: An autonomous system can have a large number of sensors. These can be configured, installed and operated in different ways. This leads to enormous complexity that needs to be mastered. Sensor groups are a concept for systematizing this problem. For this purpose, various perception tasks are identified and requirements for the sensors and their installation and configuration are derived from this. The sensors themselves are assigned to the various sensor groups and share the common properties and the perception task with the other sensors in the group. The different groups can fulfill different, possibly contradictory requirements.			

Rationale 23: *Perception Task*
The perception *Task* is initially independent of the sensor itself. It describes what information is to be perceived. There are usually several ways to fulfill the *Task*. The *Task* alone is therefore not sufficient for defining the sensor group, but it forms an important basis. The other properties of the sensor group are derived from the *Task*. However, due to the various reasons, there can be several sensor groups with the same perception *Task* on one system.

Rationale 24: *Kinematic configuration*
The kinematic configuration of the sensor group is an important design decision as to how the perception *Task* is to be fulfilled. It thus characterizes the sensor groups and strongly influences the requirements for the sensors. All sensors in a group share the kinematic configuration.

Rationale 25: *Requirements for the sensors*
The requirements for the sensors can be defined based on the *Task* and the kinematic configuration. The requirements cover a broad spectrum. For example, the sensor group can determine the size of the sensors or the working range, but also the frame rate and the infrastructure (data, power).

Rationale 26: *Reduced complexity*
By dividing the perception *Task* into *Subtasks*, the complexity can be better mastered. It also prevents the same perception *Task* from being solved unintentionally by different concepts. This avoids an unnecessary increase in system complexity. Common infrastructure requirements also reduce complexity. Simplification can also be achieved in operation by grouping sensors. For example, groups of sensors can be calibrated together.

B.3.4 Sensor Setup: Head, Hand and Navigation

Model Name:	Model Kind:	Model Type:	ID:
Sensor Setup: Head, Hand and Navigation	VP-M2A	Approach	A2

Addressed Aspects:
VP-A1, VP-A2

Description:
The sensor system of AIMM consists of four different sensor groups, which are designed for different perception *Tasks* and together fulfill the requirements of the system.
The first sensor group are the internal sensors of the mechatronic components, e.g. joint angle sensors and torque sensors.
The second group of sensors is called the sensor head. A PTU is mounted on a mast and carries various sensors.
The third group of sensors is called the hand sensor. It contains the sensors that are attached to the end effector of the manipulator.
The fourth sensor group, the navigation sensors, are mounted on the mobile platform and are used for environmental perception for navigation.

Applies:
G3, G4, G5

Rationale 27: ***Mechatronic component sensors***
The internal sensors are mainly used for the system's self-awareness. They measure the joint values, for example, and can therefore provide reliable and fast information about the status of the kinematics. Small model errors and external disturbances can thus be compensated for.
The environment can often only be detected indirectly with these sensors. External forces on the manipulator can thus be estimated, taking into account the manipulator dynamics and the built-in torque sensors.

Rationale 28: ***Sensor group head sensor***
The integration of a sensor head offers many advantages, which is probably why this approach has established itself in biology. It contains the system's most important sensors and is used to monitor the *Robot*'s workspace, perceive the environment and localize objects. As it is mounted independently of the manipulator, perception is independent of the manipulation. This means that the surroundings can also be observed during manipulation or the manipulation itself can be monitored.
For AIMM, the head sensor was implemented using a sensor mast with a pan-tilt unit. This separate installation also means that the sensors can be replaced much more easily and their size only has a minimal impact on the *Robot*'s workspace. The pan-tilt unit enables the head sensors to cover the entire working area of the *Robot* with a sufficiently high resolution to be able to detect even smaller objects. One disadvantage of this sensor group is the long kinematic chain to the end effector, which can lead to accumulated errors. Another aspect is that the head sensor can only monitor the working area through movements of the pan-tilt

unit. Simultaneous complete monitoring is not possible. A further disadvantage of the kinematics of the pan-tilt unit is that only the field of view of the sensors can be extended. Changing the direction of view of an object requires the entire system to be moved.

Rationale 29: *Sensor group hand sensor*
The *Task* of the hand sensor group is to inspect objects at close range and also to cover areas that are hidden from the head sensor. The kinematic configuration of this group is therefore to mount the sensors on the end effector. This allows the sensors to be positioned almost anywhere in the working area.

Due to the static mounting of the sensors group on the end effector, there are no accumulated errors for manipulation tasks. This group is therefore also relevant when high precision is required.

A disadvantage of this sensor group is that it is not independent of the manipulator. Perception with this sensor group is therefore difficult during manipulation. Another factor is that objects at the end effector strongly influence the workspace of the *Robot*. The sensors must therefore be small and integrated as compactly as possible. This restricts the type of sensors and makes it difficult to replace components.

Rationale 30: *Sensor group navigation sensor*
The *Robot*'s moving platform is holonomic. It can therefore move in any direction at any time. The navigation sensor group has the *Task* of fully monitoring the *Robot*'s surroundings for the platform movements. The requirement for the sensors is therefore to provide a 360° all-round view of the system. As the entire area is covered, these sensors can be mounted statically.

B.3.5 Roboception rc_visard 65

Model Name:	Model Kind:	Model Type:	ID:
Roboception rc visard 65	VP-M2I	Implementation	I2

Addressed Aspects:
VP-A2

Description:
The rc_visard 65 stereo camera system from Roboception is used as the main sensor for the sensor head. The technical specifications of this purchased component meet the requirements of the AIMM system. The software integrated in the sensor, which offers calibration and stereo processing, among other things, reduces the system complexity of the AIMM system.

Implements:
A2

B.3.6 LED-Projector

Model Name:	Model Kind:	Model Type:	ID:
LED-Projector	VP-M2I	Implementation	I3
Addressed Aspects: VP-A2			
Description: An LED projector is used to support the system's stereo sensor technology. This projects a pattern so that the stereo algorithm can also deliver good results on untextured surfaces. The pattern covers the field of view of the stereo cameras.			
Implements: A2			

Rationale 31: ***Activate and deactivate***
The pattern improves the depth perception of the system for poorly structured surfaces. However, the projected structure is a drawback for other perception *Tasks* such as detection or modeling. It is therefore important that the projector can be controlled by the system and is only activated when required.

Rationale 32: ***No calibration necessary***
There are also approaches to realize the depth perception by pattern projection with only one camera, e.g. Primesense Kinect. However, this has some disadvantages compared to the stereo system with projector. For example, the accuracy is poorer, especially for relevant features such as edges and corners. Another problem is daylight. If this outshines the projector image, depth measurement is not possible. Although a stereo system can then no longer benefit from the projector, it can still deliver measured values. The additional projector also does not need to be calibrated to the stereo system, which significantly reduces the integration effort.

Rationale 33: ***Speckle effect***
Many pattern projectors use laser diodes to generate the required light. This works for mono camera systems. However, the speckle effect plays a significant role in stereo systems. Minimal unevenness on a surface causes the coherent light to overlap differently depending on the viewing angle. The projected points are therefore perceived differently by the two cameras, which impairs the stereo algorithm. The AIMM system therefore uses an LED light source with non-coherent light.

B.3.7 Amtec Pan Tilt Unit

Model Name:	Model Kind:	Model Type:	ID:
Amtec Pan Tilt Unit	VP-M2I	Implementation	I4
Addressed Aspects: VP-A1, VP-A2			
Description: The Amtec pan/tilt unit from 2004 is the oldest hardware component of the AIMM system. It was already used in the same function in a predecessor system, the Robutler [56]. As it still meets the requirements for the sensor head, it will continue to be used on AIMM.			
Implements: A2			

B.4 AIMM Physical IT

B.4.1 Local Computing

Model Name:	Model Kind:	Model Type:	ID:
Local Computing	VP-M3G	Guideline	G6
Addressed Aspects: VP-A3			
Description: A mobile system can change its location. In order to have a reliable data connection at all times, a high technical effort is required. The concept of local computing therefore ensures that all computations are carried out on the system itself. This significantly reduces the infrastructure requirements.			

Rationale 34: ***Autarchy***
Local processing of the system achieves a higher degree of self-sufficiency. This means that the system is less dependent on its environment. This reduces the integration effort and increases the flexibility of the system.

Rationale 35: ***Clear system boundaries***
When computations are outsourced to external hardware, it is more difficult to clearly define the system boundaries. One possibility is to consider the external hardware as part of the system. This approach is supported by the fact that these external computations are an essential part of the *Task* fulfillment. However, the system boundary is then difficult to

define, especially for server-based solutions. An alternative is to define the boundary via the hardware. However, this means that the *Robot* can no longer perform its *Tasks* independently. In addition, very different computations can be outsourced. From stereo data processing to knowledge-based database queries. The interface is therefore arbitrarily complex.

The concept of local computation means that the system boundary of the *Robot* is clearly defined, as all components are physically integrated in the device.

Rationale 36: *High data bandwidth*
Robot systems generate large amounts of data. This data must be transferred from the sensors to the computers for processing. This can be done by cable within the system which allows high bandwidth. Wireless technology must be used for external transmission. However, the data rates involved cannot be fully transmitted using current technology.

Rationale 37: *Reduced efficiency*
One disadvantage of local computation is that the entire computing resources must be kept on the system. However, the full capacity is often only required for relatively short periods of time. The efficiency compared to an external solution is therefore lower.

Rationale 38: *Requirements for installation space and energy supply*
The computers on the system require space and energy. Both are limited resources on a mobile system. This technical restriction limits the amount of computing power available locally on a mobile system.

B.4.2 Unlimited Computational Resources

Model Name:	Model Kind:	Model Type:	ID:
Unlimited Computational Resources	VP-M3G	Guideline	G7
Addressed Aspects: VP-A3			
Description: Sufficient computing resources are available for all processing operations on the system. Optimization in terms of computing resources is therefore not necessary.			

Rationale 39: *Energy consumption is not a research topic of the platform*
In contrast to the practical use of a system in industry, the costs for the optimization of algorithms and modules are significantly higher in research than for the integration of additional computing hardware. As the efficient use of computing resources is also not a research topic for the group, the computing power is adapted to the requirements.

B.4.3 Processing

Model Name:	Model Kind:	Model Type:	ID:
Processing	VP-M3A	Approach	A3

Addressed Aspects:
VP-A2, VP-A3

Description:
The AIMM's IT equipment consists of four computers that perform different tasks.
There is a server computer. The task of the server is to simplify the administration of the system. All computers mount the data hosted by the server as a home directory. This implicitly ensures synchronization between the computers.
The sensor computer is the interface to all sensors. All sensor data is routed via this computer and is provided with a time stamp here if the sensor does not offer this itself.
The GPU computer is used for GPU-based data processing. This computer is equipped with a powerful graphics card. All modules that use a GPU for processing are started on this computer.
The processing computer is used to carry out computationally intensive processing. All computing-intensive modules of the system are started here.

Applies:
G4, G6, G7

Rationale 40: ***Extendability by multi computer setup***
By using several computers, it is possible to react flexibly to the demand for computing resources. Additional computers can be easily integrated into the system as required.

Rationale 41: ***Embedded computer***
There are two reasons for using embedded computers. Compared to server computers, for example, the ratio between size, power consumption and computing power is more favorable for embedded computers. In addition, this hardware is suitable for the mechanical stress caused by the movement of the mobile platform.

Rationale 42: ***Increased efficiency by specialization***
IT can be made more efficient by specializing the hardware. For example, not every computer needs a powerful graphics card. When a resource limit is reached, the necessary expansion can be carried out on a dedicated basis.

B.4.4 Network

Model Name:	Model Kind:	Model Type:	ID:
Network	VP-M3A	Approach	A4

Addressed Aspects:
VP-A3

Description:
This model describes the concepts for data transmission on the AIMM system. The main technology is ethernet and is used for various tasks. The network setup therefore consists of three independent networks:
The data network is used to connect the various computers. This network is used, for example, to implement the shared file system.
The service network is used to transport application data. Commands, for example, are transmitted to the modules via this network.
The sensor network is used to connect the sensors. In contrast to the other two network types, there is not just one connected sensor network, but several. A participant in a sensor network is always the sensor PC. Only sensor data is transmitted in these networks.

Applies:
G6, G7

The internal communication, shown in Figure B.2, is realized via ethernet connections. Each computer has two ethernet connections that form a network with all other computers. The data network is used to share a file system that is hosted on the server computer. The service network, in which the wireless access point is also integrated, is used for communication between the processes. The sensor data is processed by the sensor PC, which is equipped with four additional ports to provide the required bandwidth.

Rationale 43: ***Ethernet as communication solution***
Ethernet is mainly used on AIMM, as it offers many advantages over other technologies. For example, high data rates can be transmitted. Ethernet networks can be easily expanded using switches, which is particularly important for flexible sensor systems. The same technology can also be used for sensor connection, computer networking and external connection and control.

Rationale 44: ***Multiple sensor networks***
Some of the AIMM sensors have their own networks, i.e. a direct connection from the sensor to the sensor PC. The advantage of this configuration is that the network can be configured according to the data that is generated. For example, different settings make sense for high-frequency but small data packets than for large, low-frequency data packets such as camera data. The division into different networks also enables simpler resource management, as the data rate for the connected sensors can be fully utilized.

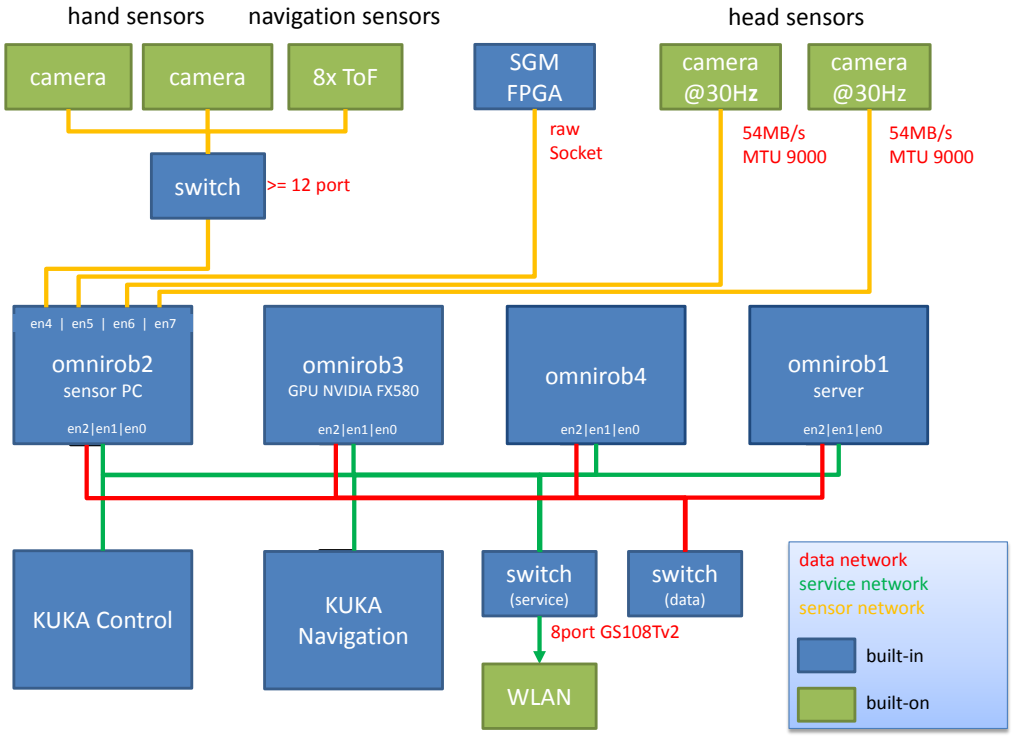

Figure B.2: *IT structure of the AIMM system. (source: Dömel et al. [35])*

B.4.5 COM Express Boards

Model Name: IT-Hardware	Model Kind: VP-M3I	Model Type: Implementation	ID: I5
Addressed Aspects: VP-A3			
Description: COM express boards are used as computers for the AIMM system. These offer a number of advantages over other solutions. In particular, their standardized form factor makes it easy to replace the hardware. As the processor is located on a separate, replaceable module, the computing power can be adapted with minimal effort. The modules are also available in the industry standard and have a large number of interfaces.			
Implements: A3, A4			

Rationale 45: ***Low energy consumption***

COM express modules can be equipped with processors for mobile use. Compared to processors developed for use in servers, for example, these have significantly lower energy consumption.

Rationale 46: ***Compact integration***
COM express modules are supplied without housing, power supply unit or display. These components are not required in the *Robot*. The integration of a COM express module is therefore much more compact than that of a laptop, for example.

Rationale 47: ***Large number of interfaces***
There are COM express modules with a large number of interfaces. The Kontron modules used on the AIMM system, for example, already offer 3 Ethernet ports as well as a large number of USB and other interfaces. This means that many devices can be connected directly to the computer, especially on the sensor PC.

B.5 AIMM Physical Power

B.5.1 Unlimited Power Resources

Model Name:	Model Kind:	Model Type:	ID:
Unlimited Power Resources	VP-M4G	Guideline	G8
Addressed Aspects: VP-A4			
Description: The system always has sufficient energy resources available. It is therefore not necessary to optimize the energy consumption.			

Rationale 48: ***Energy efficiency is not a research topic on the AIMM platform***
The running time of the system without charging plays a subordinate role for use as a research platform. As the energy efficiency of the robot is not in the group's research interests, unlimited energy resources are assumed for AIMM.

B.5.2 Power Distribution

Model Name:	Model Kind:	Model Type:	ID:
Power Distribution	VP-M4A	Approach	A5
Addressed Aspects: VP-A4			
Description: The AIMM's power supply is based on a 48V battery integrated into the system. The loads are connected to this battery via various power supply units. The system offers 5V, 12V, 24V and 48V. The system battery can be charged during operation. If the robot is connected to the grid via the charger, the consumers are supplied directly from the charger.			
Applies: G8			

Rationale 49: ***Plugged operation***
An important requirement for the power setup is power supplied operation. During integration or system tests, the system should not run on battery power, but can be connected to the charger. During this time, the *Robot* must be able to move. The risk here is that energy, e.g. from braking the platform, is fed back into the charger. This leads to damage and must be avoided.

B.5.3 KUKA omniRob Power

Model Name:	Model Kind:	Model Type:	ID:
KUKA omniRob Power	VP-M4I	Implementation	I6
Addressed Aspects: VP-A4			
Description: The battery of the KUKA omniRob is designed for an 8-hour shift without recharging. This means that the platform offers sufficient capacity even with additional consumers such as sensors and computers. In addition, the platform can also be charged when active.			
Implements: A5			

B.6 AIMM Physical Safety

B.6.1 Safety Operator

Model Name:	Model Kind:	Model Type:	ID:
Safety Operator	VP-M5G	Guideline	G9
Addressed Aspects: VP-A5			
Description: The safety of the system is the responsibility of a safety operator who monitors the system during operation. This person must ensure that no persons are harmed and that neither the system nor its surroundings are damaged.			

Rationale 50: ***Open research topic***
Currently, there are no applicable solutions to ensure the safety of autonomous mobile manipulators in a partially unknown environment. There are various approaches working towards this goal. Currently, these solutions are still under active research and cannot yet be generally applied.

Rationale 51: ***Focus on the concerns***
In contrast to the practical application of the system in industry, in research it can be ensured that a safety operator monitors the system during execution. Since the safety of the *Robot* is not in the research interest of the group, this concept was chosen for AIMM.

B.6.2 Safety

Model Name:	Model Kind:	Model Type:	ID:
Safety	VP-M5A	Approach	A6
Addressed Aspects: VP-A5			
Description: The safety equipment of the AIMM system enables the safety operator to ensure safe operation. The basis of this setup is a wireless emergency stop with which the system can be stopped and laser scanners with safety fields. If these areas are violated, the system is automatically stopped. It is also the task of the safety personnel to ensure that no persons come into direct contact with the system.			
Applies: G9			

Rationale 52: ***High latency***
A fundamental problem for a safety operator is the limited response time. At least a few hundred milliseconds are required to react to a sudden event. The safety operator guard must therefore ensure that there is always enough time to react.

Rationale 53: ***Wireless emergency stop as an interface to the safety operator***
The safety operator must have a simple and safe way to stop the system. Emergency stop switches on the system are not suitable for this, as the safety operator then has to reach into the *Robot*'s danger zone. AIMM therefore uses a wireless solution. This allows the safety operator to monitor the system from a safe distance and trigger an emergency stop. The problem with wireless emergency stop systems is that the wireless connection must be guaranteed at all times. An emergency stop must therefore be triggered automatically in the event of a fault.

Rationale 54: ***Support of the safety operator through safety areas***
The system's laser scanners can independently monitor different areas. If an object is detected in these areas, a system reaction, e.g. an emergency stop, can be triggered automatically. This approach has some limitations in the current state of the art, which is why the technology should only be seen as support for the safety operator.
The first limitation of the safety fields lies in the measuring range of the sensors, which is planar. Safety areas can therefore only be defined and monitored in the measuring plane. For example, the manipulator cannot be monitored. These sensors can therefore only be considered as an additional safeguard in case the safety operator overlooks something or reacts too slowly. Another problem is that the sensors used can only realize static safety fields. Depending on the *Task*, however, there are different requirements for the safety field. A larger safety field is desirable during navigation, as the platform has to be slowed down in an emergency. During manipulation, on the other hand, contact with the environment is necessary, which requires the platform to be positioned close to workstations. With AIMM, the safety fields are therefore reduced to a minimum to enable manipulation. The lack of automatic safeguarding of platform movements must be replaced by measures taken by the safety operator.

Rationale 55: ***No direct contact with humans***
Due to the limited reaction time of the operator and the lack of automation of the system, AIMM is not suitable for direct physical contact with the user.

B.6.3 KUKA omniRob Safety

Model Name:	Model Kind:	Model Type:	ID:
KUKA omniRob Safety	VP-M5I	Implementation	I7
Addressed Aspects: VP-A5			
Description: In addition to emergency stop switches on the robot, the KUKA omniRob has a wireless emergency stop and safety laser scanner. Both are designed using safe technology. For autonomous manipulation, however, the safety fields of the lasers must be reduced to a minimum. Safe operation without monitoring is then no longer possible. For AIMM, the wireless emergency stop is sufficient to implement the safety concepts.			
Implements: A6			

B.7 AIMM Physical Interface

B.7.1 Simple Status Display

Model Name:	Model Kind:	Model Type:	ID:
Simple Status Display	VP-M6G	Guideline	G10
Addressed Aspects: VP-A6			
Description: An autonomous system acts by itself in its environment. A complex user interface for this system is therefore not necessary. However, as people are also present in the *Robot*'s environment, it is still important to display the current status of the system. This interface should be as simple and intuitive as possible.			

Rationale 56: ***Relevant operating modes***
The most important information that the *Robot* has to communicate to its environment is whether the system is active. Since the surrounding environment is not safe, it must be clearly recognizable when the system is in operation. It is very helpful for monitoring the system if the normal state can be distinguished from the error state.

B.7.2 Low Level Motion Commands

Model Name:	Model Kind:	Model Type:	ID:
Low Level Motion Commands	VP-M6G	Guideline	G11
Addressed Aspects: VP-A6			
Description: The operator should also be able to move the *Robot* directly. The AIMM system therefore provides a low-level motion command interface with which movements can be controlled directly.			

Rationale 57: ***Need for a direct interface to move the platform***
This interface is not required in standard operation. The *Robot* decides for itself which action it performs when and how. Outside of this operating mode, the platform must also be moved. This can be used to solve error situations. When setting up the system, manual input is also initially required to move the *Robot* to its operating location.

B.7.3 LED Status Display

Model Name:	Model Kind:	Model Type:	ID:
LED Status Display	VP-M6A	Approach	A7
Addressed Aspects: VP-A6			
Description: To achieve a simple status display, AIMM uses LED strips that can be color-controlled. AIMM distinguishes between three different states. If the system is inactive, the light strips are switched off. If the system is active and in the normal state, the strips light up blue. In case of a fault, the light strips change to red. In order to be able to differentiate between different faults, the strip is divided into segments to signal different fault states. For example, a red illuminated bumper indicates an unacknowledged emergency stop.			
Applies: G10			

B.7.4 Remote Controller

Model Name:	Model Kind:	Model Type:	ID:
Remote Controller	VP-M6A	Approach	A8
Addressed Aspects: VP-A6			
Description: A remote control is used to command the movements. It allows the specification of speed values for the robot movement.			
Applies: G11			

Rationale 58: ***Kinematic reduction***
Only the three degrees of freedom of AIMM's platform can be controlled with the remote control. This is sufficient to reposition the *Robot*. The manipulator cannot be moved remotely, as this would require a more complex input device.

B.7.5 KUKA omniRob Remote Control

Model Name:	Model Kind:	Model Type:	ID:
KUKA omniRob Remote Control	VP-M6I	Implementation	I8
Addressed Aspects: VP-A6			
Description: The KUKA omniRob is supplied with a remote control. When the *Robot* is activated, the system automatically switches to a state that allows the platform to be controlled directly via a gamepad. However, it is not possible to control the manipulator. For AIMM, this is sufficient to realize the operating concept.			
Implements: A8			

Appendix C

AIMM Capability View

This chapter describes the *Capability View* of the AIMM system. An overview of all identified *Models* and their interrelationships is provided first. The individual *Architecture Models* are then presented.

C.1 AIMM Capability Overview

***Guideline Models*:**

- Hierarchical Loop Closure
- Flexible Combination of Capabilities
- Tree Structure
- Central World Model
- Virtual Objects
- Scene Concept
- Combining Middlewares
- Fine Granularity
- Distributed Dedicated System
- Central Module Management
- Individual Runtime Environments
- Process Dependencies

***Approach Models*:**

- Dynamic Capability Pipelines
- Robot Behaviours
- World Model
- World Model Modules
- Communication Map
- Host Types
- Process Manager

***Implementation Models*:**

- World Model Framework
- Middleware Selection
- LN Manager
- RAFCON

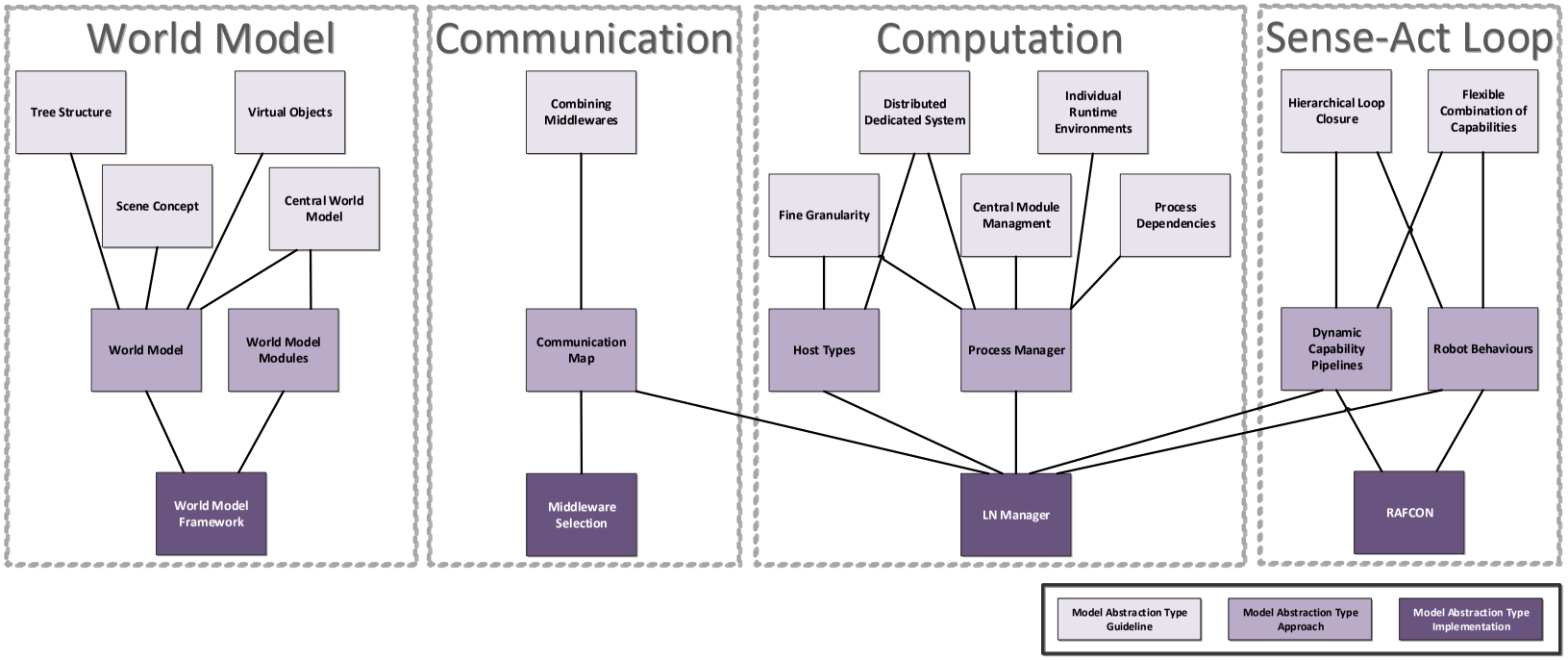

Figure C.1: *The Relations of the Capability View Models*

C.2 AIMM Capability Closing the Action Perception Loop

C.2.1 Hierarchical Loop Closure

Model Name:	Model Kind:	Model Type:	ID:
Hierarchical Loop Closure	VC-M1G	Guideline	G12
Addressed Aspects: VC-A1			
Description: The action-perception loop can be closed at different levels of abstraction. A problem can be solved by control, by reactive behavior or by planned actions. Each level has advantages and disadvantages, which is why it is not possible to commit to one level for more complex systems and *Tasks*. The concept of hierarchical loop closure is to combine the advantages of the different approaches by cascading the different levels.			

Rationale 59: *Fast controller*
A controller calculates a control value from the measured values of the sensors in order to reduce the deviation between the actual value and the setpoint. This control loop is very fast and robust, but cannot solve complex problems.

Rationale 60: *Reactive behaviors*
Similar to the controller approach, the actions are derived directly from the *Robot*'s sensor signals. In a reactive behavior, however, there is no actual and target value, but a behavior that depends on the measurement. For example, a collision avoidance behavior detects obstacles based on the depth information. Depending on the distance of the obstacles, virtual forces are generated that adapt the movement of the *Robot*. As the calculations for reactive behavior are often more complex, they are slower than controllers. Nevertheless, reactive behavior is controlled by sensor data and the reaction is usually faster than a planned action.

By linking different information and actions, more complex *Tasks* can also be solved. For example, Brooks [17] presents a mobile robot system that is able to navigate and map an unknown environment using reactive behavior.

Rationale 61: *Planned actions*
Closing the loop with a planner is abstract as there is no direct processing of the captured data into a response. The perceived data is usually used to model the environment. For example, mapping algorithms and object localization methods

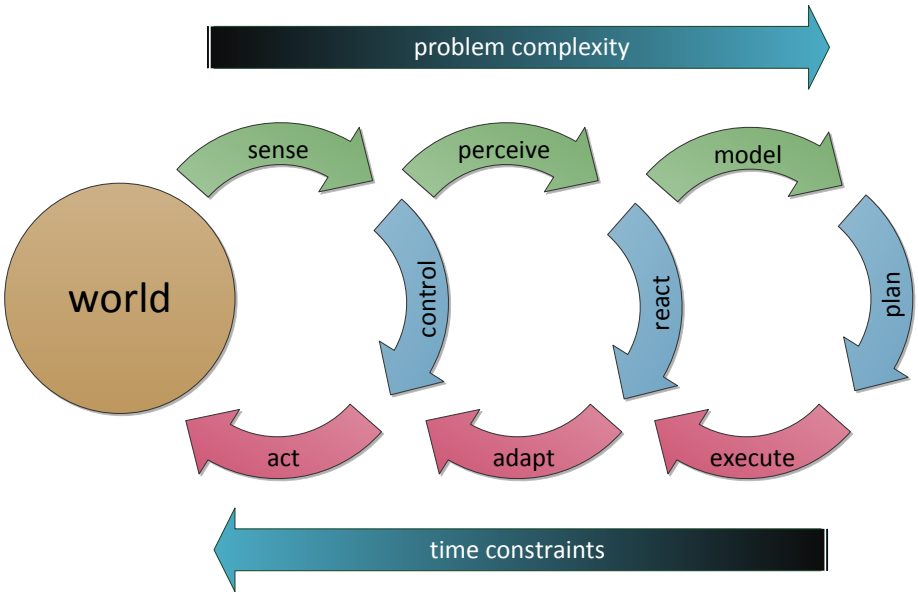

Figure C.2: *Extended sense plan paradigm closing the loop on three layers*

generate knowledge about the world that is not directly used for an action. The planner uses the existing, often accumulated knowledge to calculate a solution for a specific problem. An example of this is the calculation of a collision-free path using the map and the positions of the objects. The output of a planner is a result at a high level of abstraction, e.g. a sequence of actions.

Planners are able to use aggregated information from the past as well as consider the future effects of actions. Compared to reactive behaviors and controllers, planners are slow, but can solve complex problems.

Rationale 62: ***Hierarchical combination of approaches***
None of the individual levels can fully meet the requirements of an autonomous system. Therefore, the different approaches must be combined. The different layers are usually combined hierarchically, as shown in Figure C.2. The upper layers have no direct connection to the physical world, but use the lower layers as an interface. In this way, the loop is closed on several levels simultaneously.

For example, the collision-free path determined by the planner is used by a collision avoidance behavior that reacts to dynamic obstacles and uses an impedance controller to enable safe interaction between humans and robots.

This concept underlies many *Robot* architectures introduced in the 1990s, e.g. SSS architecture [29], 3T architecture [13], and LAAS architecture [3].

C.2.2 Flexible Combination of Capabilities

Model Name:	Model Kind:	Model Type:	ID:
Flexible Combination of Capabilities	VC-M1G	Guideline	G13
Addressed Aspects: VC-A1			
Description: The action-perception loop is realized by combining different *Capabilities*. For example, sensor data is recorded by a *Capability* and used by a control *Capability* to calculate a setpoint for the actuator *Capability*. In AIMM, the concept is to keep these combinations flexible. This means that the topology of the *Capabilities* can be changed at runtime. For example, *Capabilities* can be exchanged or new *Capabilities* can be added.			

Rationale 63: *Situative behavior*
An autonomous system has to adapt its behavior to the context. For example, a *Robot* must behave differently in an unsafe environment than in a known, safe environment. The type of uncertainty also plays a significant role. If the environment is static, strategies for scene analysis and motion planning can solve the problem. However, if the *Robot* is in a dynamic environment, reactive methods have to be used.
To achieve this variation in behavior, different *Capabilities* must be used in different combinations depending on the context.

Rationale 64: *Reducing complexity by separation*
This adaptive behavior of the system can be achieved through various approaches. One possibility is to activate all *Capabilities* simultaneously. The influence of the respective component on the overall system is then set via weightings. For simple systems, complex behaviors can be implemented in this way [17].
For complex systems, this static approach of using all *Capability* together is difficult to manage. In order to reduce complexity, this approach does not attempt to find a solution for the entire spectrum of behaviors. Instead, simpler constructs of *Capabilities* are developed, each of which realizes a subset of the required behaviour. The system is then reconfigured flexibly depending on the required solution.

Rationale 65: *Resource efficiency*
Some *Capabilities* require larger amounts of resources, e.g. in terms of working memory or processor utilization. Flexible use allows the same *Capabilities* to be used for different problems. This saves resources and increases the efficiency of the system.

C.2.3 Dynamic Capability Pipelines

Model Name:	Model Kind:	Model Type:	ID:
Dynamic Capability Pipelines	VC-M1A	Approach	A9
Addressed Aspects: VC-A3			
Description: Several *Capabilities* can be combined by connecting them in sequence. The result of one *Capability* is the input for the next *Capability*. This structure is referred to below as a pipeline. The pipeline becomes dynamic by changing the links at runtime. Hierarchical loops can be realized with pipelines by branching.			
Applies: G12, G13			

Rationale 66: *Data driven*
Capability pipelines are data-driven. The information entering the pipeline is processed sequentially by the various *Capabilities*. As soon as new information is available, the *Capability* can determine the output based on the internal model and the parameters and pass it on. Within the pipeline, the data is therefore triggered by the *Capabilities* and the parameters usually remain unchanged over several data sets.

Rationale 67: *No Capability layers*
The concept of the dynamic *Capability* pipeline does not distinguish between the different levels of abstraction of the *Capabilities*. Often, the data in a pipeline becomes more complex with increasing processing depth, but even high-level *Capabilities*, such as path planners, may require low-level input, such as the current joint configuration. It is therefore not useful to identify layers or even interfaces between different layers.

Rationale 68: *Structure of a dynamic Capability pipeline*
There is at least one data source at the start of a dynamic *Capability* pipeline. This can be a hardware interface or an event-triggered *Capability*. Within the pipeline, data triggered *Capabilities* can be interconnected as required. In order to realize the dynamics, both the interconnections and the parameters can be changed at runtime. The end of a pipeline is either a data aggregating *Capability* or a hardware interface *Capability*. It can also make technical sense to run a pipeline without terminating components. The relevant *Capabilities* are then dynamically linked to this pipeline as required.

Rationale 69: *Timestamps*
For many types of data, a timestamp is important. With pipelines, the original timestamp is usually passed on, i.e. the data does not receive the timestamp of the processing, but the timestamp of the creation of the information.

C.2.4 Robot Behaviours

Model Name:	Model Kind:	Model Type:	ID:
Robot Behaviours	VC-M1A	Approach	A10
Addressed Aspects: VC-A1			
Description: The connection of *Capabilities* implements the behavior of the *Robot* and is referred to below as *Robot* behavior. This indicates that both the actions that are executed to fulfill the *Mission* and the reactions to the environment are realized by *Capabilities* and their interconnection and parameterization. However, this behavior is adapted or exchanged at runtime in accordance with G13. Several *Robot* behaviors can also be active at the same time.			
Applies: G12, G13			

Rationale 70: *Complexity of behavior*
The complexity of *Robot* behaviors can have a very broad spectrum. Very simple behaviors can be implemented with one or two *Capabilities*. An example of such a behavior is the emergency stop switch. As soon as the switch is pressed, the brakes of the *Robot* are activated.
It becomes more complex when the input data is linked to the output data via a controller. This is of course also possible at the reactive level, the planning level and any combination thereof. There are no upper limits to the complexity of *Robot* behavior. Many traditional *Robot Architectures* only implement static robot behavior (subsumption, 3T, etc.) in order to fulfill the *Mission*. This could be used to solve complex tasks such as navigation in unknown, dynamic environments. In order to realize such a *Task* using a behavior, a large number of *Capabilities* have to be linked together. AIMM's behaviors are also modified, activated and deactivated at runtime.

Rationale 71: *Closed-Loop and Open-Loop*
There are different types of behaviors. Simple behaviors often close the sense-act

loop directly. For example, a gravity compensation behavior is based on the direct connection of measured torques with controlled torques.
However, there are also behaviors that are open-loop. This means that they do not react directly to the environment, but work without feedback. One example of this is the *Robot*'s behavior of executing a trajectory. A trajectory is calculated from the path. This is then commanded to the actuators.

Rationale 72: *Execution of robot behaviors*
Robot Behaviors can be executed in different ways. There are also different types of *Robot* behaviors in terms of execution duration. There are behaviors that are constantly active. An example of this is the emergency stop behavior. Other *Robot* behaviors can run continuously, but are explicitly started and stopped. For example, AIMM has a behavior to track the end effector with the cameras. This is only desirable if the cameras are not used for other purposes. Other *Robot* behaviors have a predefined duration or a predefined goal. These behaviors are triggered and then terminate themselves. For example, the behavior "driving a trajectory" ends when the trajectory has been completed.

C.2.5 RAFCON

Model Name:	Model Kind:	Model Type:	ID:
RAFCON	VC-M1I	Implementation	I9
Addressed Aspects: CVC3, CVC4			
Description: Dynamic pipelines and high-level *Robot* behaviors are implemented for AIMM with RAFCON. RAFCON state machines can trigger behaviors that are implemented externally. By using the data flows, RAFCON can also be used directly to implement pipelines and *Robot* behaviors.			
Implements: A9, A9			

Rationale 73: *Dynamic Capability pipelines*
RAFCON can call *Capabilities*, the returned data can be passed on to the next state via a data port. From here, the data can in turn be passed on to the next *Capability*. In this way, *Capability* pipelines can be set up directly. RAFCON can also change the

data flow during execution based on the data or with a superimposed control, which makes the pipeline dynamic.

Rationale 74: ***Robot Behaviors***
RAFCON can also link *Capabilities* cyclically. In the simplest case, the logical output of a state is switched back directly to its input. Of course, several states can also be linked cyclically and adapted dynamically. For example, a tracking behavior can be implemented by cyclically linking an object detection and a movement command. RAFCON then triggers the object recognizer, looks at the corresponding position with the movement command and then triggers the object recognizer again. More complex behaviors can also be implemented here. For example, a search strategy can be started if object recognition fails.

Rationale 75: ***Efficiency and speed***
RAFCON is not suitable for high timing requirements or large amounts of data. No control loops can be implemented and RAFCON is also not the right tool for perception pipelines due to the amount of data.

C.3 AIMM Capability World model

C.3.1 Central World Model

Model Name:	Model Kind:	Model Type:	ID:
Central World Model	VC-M2G	Guideline	G14
Addressed Aspects: VC-A2			
Description: The world model is the central knowledge representation of the AIMM system, which stores all the knowledge and information that the *Robot* has about its world (*Robot* and environment). The information for the various skills is extracted and synchronized from this central model. New information is continuously integrated into the world model. The aim of the world model is to represent the entire world of the *Robot* at the selected level of abstraction.			

Rationale 76: ***World models are required for complex systems***
Autonomous *Robots* can also exist without an explicit representation of the world.

For example, the *Architecture* by Brooks [17] is based on behaviors that are direct reactions to sensor measurements. *Tasks* such as exploration and obstacle avoidance could be solved with this approach.
For more complex *Tasks*, such as pick-up and delivery services, this approach has not yet been successful. There are several reasons for this. Many methods require more complex models of the environment. For example, a path planner usually requires a geometric model of the environment. The choice of methods is therefore limited if no central world model is created. Another problem is that the complexity of these systems is difficult to master. Since the sensor signals are processed directly, all methods are directly coupled to these signals. The actuator signals in turn result from a weighted sum of the respective methods. If different behaviors are to be implemented, the approach requires a complex system for weighting the methods used in parallel. Isolating errors or optimizing behavior is difficult due to the lack of modularization.
A central world model is therefore necessary for autonomous robots that perform more complex *Tasks*.

Rationale 77: *Explicit modeling of dependencies between components*
In recent years, the development of *Robots* has benefited greatly from the design concept of a finely granular, modular approach. As a rule, more than a hundred different modules are used in parallel in today's *Robots*. One design goal is to decouple these components. So why develop a central world model?

The modularization of components comes from software development and is essentially based on the definition of interfaces between the modules. From a software perspective, the components are therefore independent, as information is only exchanged via the interfaces. However, most modules of a *Robot* have a relation to the physical world. As shown in Figure C.3, the modules are dependent on this.

For example, if component A is an object localization *Capability*, the result of this *Capability* depends on the physical position of the object. Other *Capabilities* also depend on this position, e.g. *Capability* B could be a grasping *Capability*. This dependency is not directly visible within the system, as it leaves the system boundary. From the system perspective, the object recognizer provides an evaluation of the transferred camera image and the gripping *Capability* closes the gripper at the transferred pose. The fact that both depend on the same physical object is theoretically irrelevant. The internal dependency can be modeled by the information flow. For example, the grasping *Capability* depends on the object recognition *Capability*. Or the implicit dependency is simply ignored.

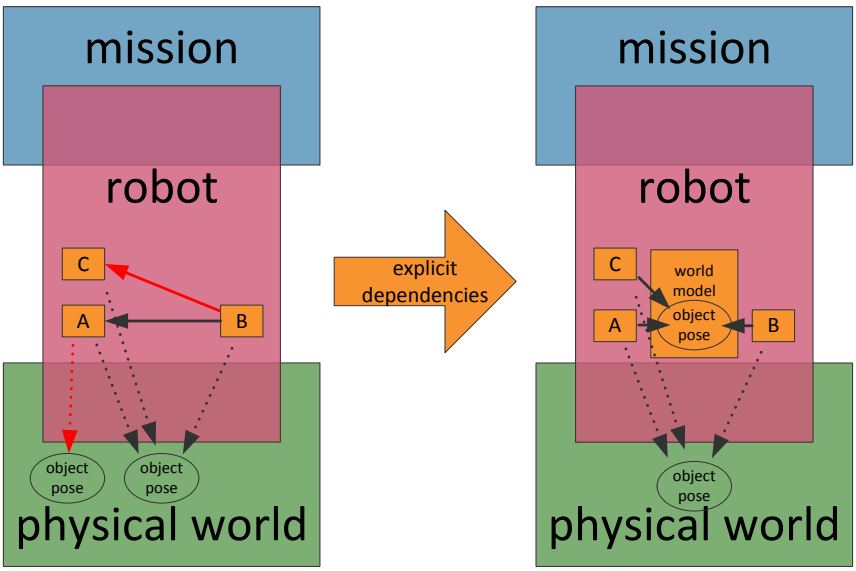

Figure C.3: *Cross-system dependencies on Capabilities and the resulting dependencies and problems (red arrows). These dependencies can be explicitly modeled within the system using a central world model.*

Both variants are problematic for an autonomous system. The artificial dependency via the data flow firmly links the components to each other. If component A fails, component B cannot be used either, even though component C could provide the required information. If, on the other hand, the dependencies are simply ignored, information can also be lost. For example, if components A and C recognize the object at different positions, at least one piece of information is incorrect. With the help of a world model, the external dependencies of the system can be represented explicitly. This provides a representation of the object and the dependencies of the various components on this object. These correspond to the dependencies on the real object that exists outside the system boundaries. A central world model therefore does not create any new dependencies between modules, but explicitly models the existing dependencies between the various modules.

For technical reasons, however, it is usually not possible to avoid internal world models in the modules. The resulting structure is shown as an example in Figure C.4.

Rationale 78: ***The real environment has exactly one state that can only be captured partially***

Another aspect that supports the concept of the central world model is that the real

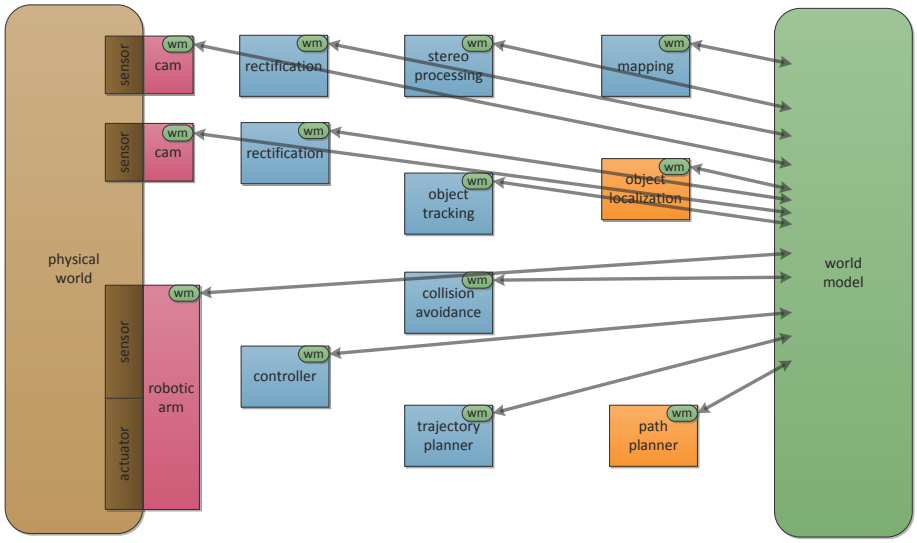

Figure C.4: *Exemplary representation of an autonomous robot system with distributed modules and their relationship to the central world model.*

world only has one state. However, this is neither fully known nor measurable for the autonomous *Robot*.

Robots receive information about their environment from various sources beyond their system boundaries. This can be, for example, user input or data such as CAD models on the IT interface or sensor data from the physical world. However, all information received by the system may be incorrect or contain inaccuracies. In addition, only in a few cases, e.g. reactive behavior or control, is the measurement data is used directly. In most cases, the information obtained must be interpreted in order to be useful for the operation of the *Robot*. An important basis for interpretation is the *Robot*'s knowledge of the world.

A very elementary knowledge for the interpretation of the information is that the environment has exactly one state. Information from different sources and points in time must therefore fit together. The central world model is a way of modeling this knowledge. Just like the real environments, the central world model therefore has exactly one state. This state should be identical to the state of the real environment.

If new information enters the system via the system boundaries, it must be checked for consistency with the world model. Based on the existing and new information, the system can then determine the most probable world state. This is then saved as a

new state in the world model.

C.3.2 Tree Structure

Model Name:	Model Kind:	Model Type:	ID:
Tree Structure	VC-M2G	Guideline	G15
Addressed Aspects: VC-A2			
Description: The world model of the AIMM platform focuses on the physical environment. The most important knowledge about this environment is the laws of physics. In order to approximate these, the world is modeled by objects that are related to each other via a tree structure, i.e. each object has exactly one superordinate object.			

Rationale 79: ***Information loss by modeling***
In general, it can be said that information is lost during modeling. The world model is therefore always a simplified representation of the *Robot* and its environment. However, this can also be used deliberately to reduce the complexity of the real environment through abstraction. In addition, not all aspects can be fully captured in the real application. World models therefore usually focus on certain areas. These two relationships are shown in Figure C.5.

Rationale 80: ***The focus is on the physical world***
As shown in Figure C.5, world models can model different areas of a *Robot* and its environment. For the AIMM system, an abstracted tree structure was chosen as the world model, which is marked in red in the figure. As described in Appendix A, the focus and research interest of AIMM is on how a system with incomplete knowledge can act autonomously in an industrial environment. A model of the physical world is therefore fundamental to AIMM.
In contrast, the *Tasks* of the system in an industrial context is very well defined. There is no need to generate missing information or explore different solutions. An explicit world model for the *Mission* is therefore not used in the AIMM.
The world model is therefore strongly focused on the physical world.

Rationale 81: ***Selection of the Tree-based World Model Structure***
As shown in Figure C.5, there are different world model structures. These are suitable for different aspects and applications. In the following, some well-known world model structures are examined for their suitability for the AIMM platform:

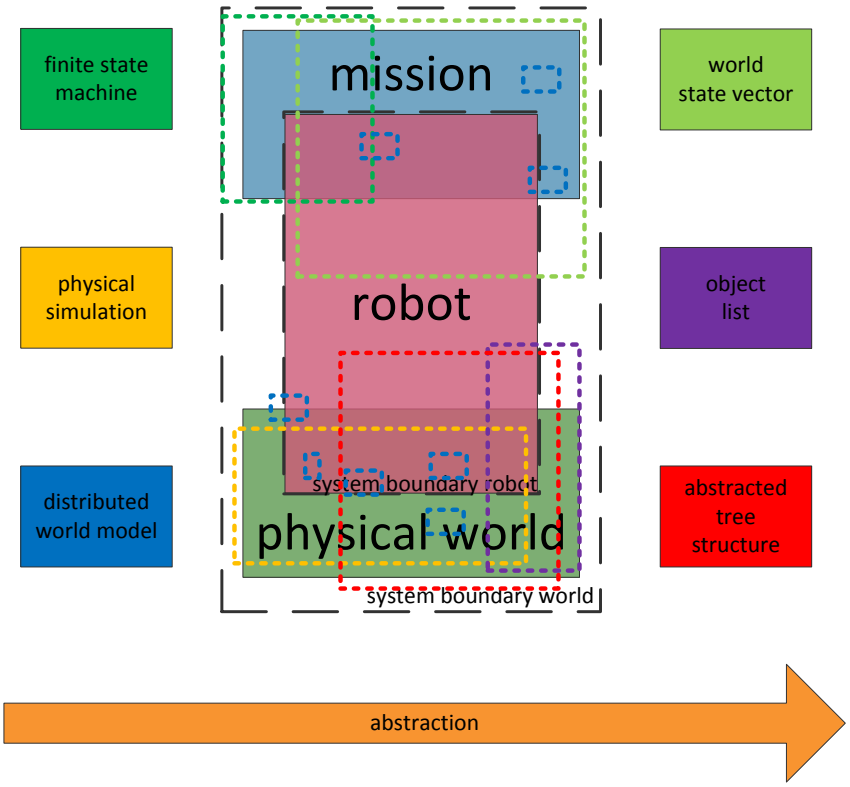

Figure C.5: *Illustration of different world model types with regard to their degree of abstraction and focus*

Finite State Machine A finite state machine can be used as a world model for a *Robot*. There is a finite number of states in this structure. The system is always in exactly one state. Events trigger transitions from one state to another.
This structure is well suited to implementing the world model of a vending machine. Due to the limited possibilities for action and the limited interaction with the environment, such machines can be represented completely and clearly by state machines.
This approach is not suitable for an autonomous *Robots* with uncertainties in the environment, a wide range of possible applications and a continuous physical environment. The complete modeling of the world of such a system would require countless states and transitions. In the field of robotics, finite state machines that focus on a sub-area of the *Mission* are suitable. For example, higher-level user interactions are modeled by finite state machines. However, this structure is not suitable as a world model for AIMM, as the focus is on the physical environment.

World State Vector The World State Vector is a structure that overcomes the limitations of the Finite State Machine through two mechanisms. The first mechanism is to define the state in multiple dimensions. This state vector can consist of any number of dimensions. These dimensions can also be continuous. The second mechanism is that transitions are not bound to a state. Instead, there are actions that change the state vector in a defined way. The same action can therefore be applied to different states.
These mechanisms extend the power of the world model enormously. By dividing the state into different dimensions, much more complex systems can be captured. It can also be modeled very well that different state dimensions are independent of each other.
By separating state and action, *Tasks* can be represented very compactly. A *Task* is defined as the desired state of the world. The system can then independently determine how the current state can be transformed into the target state using the available actions.
Symbolic planners such as STRIPS [42] are often used to determine the sequence of actions. The basis for these planners is the world model in the form of a state vector and the set of actions with their effects on the vector. The planner can then virtually apply the various actions to change the current state so that the target state is achieved. If a sequence of actions is found that achieves the goal, it can be executed, leading to the fulfillment of the *Mission*. This approach can also be used to achieve a reaction of the system to unforeseen events: The event changes the world state vector. The plan can then be adapted to this new state or a new plan can be created. A *Robot* with such an approach can therefore achieve a high degree of autonomy.

However, the world state vector approach with actions based on it also entails a number of limitations:

One problem is finding the right dimension and granularity of the state vector. For example, the state vector must contain everything that concerns the *Robot* and its environment for full coverage. In addition, the vector must also contain the *Mission* state. There are therefore very different states in this vector.
For example, a *Robot* deployed in a disaster scenario has one dimension for "joint angle", another dimension for "region is secured". Changing these completely different states with one set of actions is problematic because they are not independent of each other. For example, the *Robot* will change its joint angles to explore a region. When the region is fully explored, the region secured state can be set. There must be an action, e.g. secure region, which changes the state of the region from unsecured to secured. However, this action requires that there is not only a change of state in the joint angle, but an entire trajectory. This in turn does not fit into the abstraction of actions that are only described by a change of state. The necessary hierarchies are missing in the state vector model.

Another problem is the lack of a temporal component. The state vector is discrete-time, i.e. an action changes the state in a defined way. How exactly the state change takes place within the action is usually not defined. The state vector is therefore only actually known between the actions. This restriction can be accepted for a sequence of actions. However, if actions are executed simultaneously, the world state is in an undefined state after one of these actions is completed.

Another limitation is that the world state model with action set is only based on active state changes, in which the world state is only changed by actions. A change of the world state from the outside is not intended and cannot be modeled directly. Reactive behavior of the form if A happens then do B can only be integrated indirectly by rescheduling. Due to the deterministic model, there are also no decisions that are made at runtime, as no new information is formally generated during execution.
Another problem is that inaccuracies and incomplete knowledge are difficult to integrate into the state vector. If a state is not known, it cannot be changed by actions. If the action does not lead to a specific state change, the world state cannot be adapted correctly. Of course, all possible variants can be managed as potential world states.

In practice, however, this quickly leads to an unmanageable number of state vectors to be managed in parallel.

In summary, it can be said about the world model World State Vector that it can

be a very powerful model in combination with the modeled actions. Especially for complex tasks for which a correct sequencing must be found, such a world model can lead to a high degree of autonomy. However, the model is unsuitable for the physical environment, as simultaneities, external events and incomplete and inaccurate knowledge are omnipresent. This world model is therefore not used for AIMM.

Physical simulation In terms of its approach, physical simulation is the opposite of the state vector of the world. There is no state vector that can be discretely changed by actions, but the world behaves continuously according to the laws of physics. The *Robot* can exert forces, but it can also absorb them. The resulting world state results from the accumulated forces. The focus of this model is naturally on the physical environment.

Both the actions of the *Robot* on its environment and the influences of the environment on the *Robot* can be modeled. The additional knowledge can be used to generate information that goes beyond the measured information. For example, the *Robot* cannot measure whether something falls down, but thanks to the physical model it knows that objects do not float in free space. Based on this knowledge, measurement errors can also be recognized and corrected. The model can also correctly model dynamic environments and extrapolate future states of the world.

These advantages are accompanied by some disadvantages. For example, the world model is limited to the physical world. A complete modeling of environments including the mission state is not possible. Since the focus of AIMM is on the physical world, this can be accepted. Much more critical is the fact that the physical simulation requires much information in order to deliver realistic results. On the one hand, this concerns the properties of the environment such as masses, friction coefficients, geometries, stiffnesses, etc. and, on the other hand, its exact state such as speeds, positions, forces, etc. Small errors and inaccuracies can lead to completely different states of the world, especially in contact situations. The physical simulation must therefore be constantly checked for plausibility. The aim of AIMM is to be able to interact with any objects in unknown environments. The information required for a physical simulation is therefore missing and cannot be determined with sufficient accuracy using the system's sensors. Another disadvantage of physical simulation is the enormous computational effort. Physical simulation is therefore out of the question as a world model for AIMM.

Object List In list-based world models, the world consists of various objects. These objects can have different properties such as geometry, mass, etc. All objects have the

property of being located in a common coordinate system in space. The objects in this world model structure are completely independent of each other. All modifications to the world must be carried out explicitly, e.g. the position property of a moving object must be constantly updated. One advantage of this abstract representation of the world is that very heterogeneous knowledge can be stored. As long as information can be assigned to an object, it is possible to integrate it into the world model. Another advantage of this very abstract model is that it is easy to implement and requires hardly any computing resources. Additionally the high degree of abstraction makes it easy to represent even complex environments.

Due to the high degree of abstraction, however, the model is also subject to some limitations. One major limitation is the lack of relationships between objects. In the physical world, on the other hand, objects are always related to each other.

In practical application on a *Robot*, this leads to a number of problems. For example, physical objects can be contained within others. The screws in a container remain physically in the container when it is moved. This relationship must be explicitly monitored in the object list structure. The object position of each screw must therefore be constantly updated. Similar dependencies occur when the *Robot* grasps an object. The movement of the *Robot*'s end effector changes the position of the object in the real world, but the world model must also be told on which trajectory the object is moving. Depending on the environment, this tracking is very time-consuming and a constant source of potential errors.

Another problem is that the position of objects in world coordinates cannot be measured, as it is only an abstract definition. In order to estimate the position of an object, a local reference coordinate system is required. This can be the coordinates of the camera system, but also other objects. If the global pose of the reference coordinate system is known, the global position of the object can be calculated. The same reference object or the same reference coordinate system is often used for many objects. However, the position of the reference object is subject to errors. If this position can now be better determined, there are two ways of dealing with this information. The simplest approach is not to use it. The knowledge of the error is ignored. All positions that are determined on the basis of this reference are and remain subject to errors. The alternative is to keep track of which pose was determined with which reference. With an improved reference, the position of all associated objects can then be adjusted. This variant is also not optimal, as artificial movements are generated even though the position of the objects has not physically changed.

These two world modifications occur very frequently on AIMM. A different world

model structure is therefore required.

Abstracted Tree Structure In terms of its approach, the abstract tree structure can be classified between the physical simulation and the object list. Like the object list, this structure of the world model is very abstract. As with the object list, the world model consists of objects that can have different properties. In addition, all objects have the property pose in space. However, this does not refer to a world coordinate system, but to a superordinate object.

The resulting tree structure is used to model the physical dependencies of the object relationships and also the paradigm of the local reference objects. Each object has exactly one parent object with a defined transformation. Changing this transformation implicitly leads to changes in the positions of all sub-objects in relation to the world coordinates.

For our world model, a basic physical understanding of the world is therefore represented by the tree structure: There are objects in the world. These are geometrically related to each other. Instead of simulating physical laws, we assume that every transformation is constant until we receive new information. Moving an object moves other objects relative to the world coordinate system if they are sub-objects. Although the physical simplifications are massive, the model is sufficient to enable the *Robot* to predict and understand the state of the world beyond pure measurements. This is especially true for simple tasks, such as grasping and transporting, in a quasi-static environment. For example, when the *Robot* recognizes a known object in the scene, the position of the object is usually measured in camera coordinates. However, the physical relationship to the table on which the object is placed is much more relevant than to the cameras mounted on the *Robot*, which move in the next time step and change the relationship to the object. Due to the structure of the world model, the *Robot* knows that the object is a child of the table and therefore the position of the object relative to the table is stored. A subsequent movement of the cameras therefore has no influence on the position of the object relative to the table.

If a reference object is more precisely measured or moved, the global position of all child objects changes automatically without the local transformations having to be changed.

Nevertheless, the tree structured world model is still a very abstract and simple representation of the world, which leads to some limitations when modeling real world effects. One limitation is that only one parent is allowed. For example, the relationship of an object lying on top of two other objects cannot be represented by the tree. The merging of two objects and the resulting new object cannot be modeled

explicitly either. The assumption of a quasi-static environment naturally leads to problems if the environment is dynamic, especially if these objects interact. Another problem is that the reasons for the static transformation are not modeled and are all interpreted in the same way, at least at the top level: always constant. For the model, there is no difference between an object that is next to the reference object, an object that contains another object and an object that is mounted on another object.

Despite these limitations, the world model has proven to be useful for AIMM. An abstract, rather simple world model was deliberately chosen as this reduces complexity. Of course, the tree structure is only a very rough model and only useful for some aspects, but it is sufficient for many use cases. To a certain extent, it even makes sense to have such strong constraints, as it is much easier to estimate what information can and cannot be extracted from the model. With more complex models that attempt to simulate the real world as realistically as possible, the question always arises as to whether and to what extent the world model is reliable.
Another problem is that more complex models require more information, most of which is very difficult to capture with sensors. Therefore, a lot of expert knowledge about the environment has to be generated and passed on to the system, which practically reduces autonomy. For AIMM therefore a Abstracted Tree Structure World model is chosen, which excludes some use cases:

- Evaluation and optimization of the world state with the help of physical laws, e.g. by simulating gravity, inertia or friction
- Simulation of the real world at sensor and actuator level for testing the *Robot*'s software components
- Modeling of task states that are not implied by geometry and object states
- Additional relationships between objects, e.g. parts of a composite object

C.3.3 Virtual Objects

Model Name:	Model Kind:	Model Type:	ID:
Virtual Objects	VC-M2G	Guideline	G16
Addressed Aspects: VC-A2			
Description: The concept of virtual objects complements the concepts of the central world model and the abstract tree structure. Despite the focus on the physical world, the system generates and records information that is not of a physical nature but has a geometric reference to objects. This information can be very different. It ranges from simple geometric information such as grasp positions and camera angles to semantic objects. Through abstraction, this information can be stored in the central world model. The world model therefore also serves as a knowledge base beyond the physical world.			

Rationale 82: ***Extention of the knowlegde representation***
Virtual objects extend the options for knowledge representation in the world model. A world model that only contains physical objects can only store knowledge about these objects. For a *Robot*, however, further information is relevant that is related to the physical objects but is not itself a physical object. For example, the position at which an object can be placed is relevant for a *Robot*. Based on the knowledge of the physical objects in the environment, the workspace of the *Robot*, the geometry and the restrictions of the object to be placed, etc., such a place can be determined. This place then has a geometric relationship to a physical object, e.g. a table, but is not itself a physical object. With the help of virtual objects, such a place object can be integrated into the world model. This makes it possible to know where an object can be placed. This enables knowledge generation and application to be decoupled.
This possibility of storing *Robot*- or *Task*-specific knowledge can be a great advantage for the system. Calculations can be parallelized, results can be reused and processes can be optimized.

Rationale 83: ***Semantic objects***
Virtual objects enable the integration of semantic objects into the world model. In the simplest case, these objects can be used to attach labels to objects. For example, all physical objects that are *edible* could be given a semantic child *edible*.

Semantic objects can also be used to group objects. As described in G15, the tree structure of the world model also has some limitations. For example, the relationship between objects is always a $1 : n$ relationship and there is only ever one relationship

between two objects. These restrictions can be mitigated with the help of semantic objects. A semantic object can be inserted between two objects that describes a further relationship between the two objects.
For example, objects can lie on a table. The objects are then stored as children of the table object. In the tree structure, however, it is no longer possible to distinguish whether the objects are on the table or under the table. This classification can be modeled with the help of semantic objects. Two semantic children are added to the table: "on" and "under". All other children of the table are then attached to these two semantic objects. The objects are still children of the table, but now second degree. However, if required, the information as to whether they are on or under the table can be extracted from the tree structure.

Another limitation is that there is only one type of relationship, the geometric one. This can be extended with semantic objects. In a purely physical tree structure, it is not possible to show that two parts are assembled into one object. Once the two parts are joined, they must be removed from the world model and a new object, the joined one, must be added. This can be avoided with a semantic "assembeled" object. When parts are assembled, they are attached as children of the semantic "assembeled" object. All operations performed on the "assembeled" object, e.g. movements, affect all children. The transformations between the children remain constant. Operations on the individual children can in turn be interpreted as decompositions.

Semantic objects can be used to increase the power of tree-like world models. However, the fundamental restriction that each object can only have one parent object cannot be circumvented. It is therefore not possible to model in the tree structure that an object lies on top of two others.

C.3.4 Scene Concept

Model Name:	Model Kind:	Model Type:	ID:
Scene Concept	VC-M2G	Guideline	G17
Addressed Aspects: VC-A2			
Description: The AIMM platform is able to move autonomously. The workspace of this *Robot* is therefore theoretically unlimited. However, the part of the environment that can be changed by the system or detected by its sensors is limited to the current direct environment. The scene concept reflects this fact in the world model. A scene is therefore the part of the world that can currently be captured and changed by the system. The *Robot* itself must always be part of the active scene.			

Rationale 84: ***Scenes in the tree structure***

There are various ways to define a scene in the world model. From a physical point of view, a geometric space can be determined on the basis of the sensors that can currently be detected. Another possibility would be to make the selection of the scene dependent on the *Task*. For example, a scene could cover the part of the world in which the *Robot* system is located during a work step.

For AIMM, the approach for the scene concept is to use the tree structure of the world model. For this purpose, one node of the tree is selected as the scene root. All children of this node are then part of the scene. This approach has the advantage over the other options that it can be very easily integrated into the world model. No complex calculations and analyses are required to determine which objects belong to a scene. Nevertheless, the nature of the world model ensures that a local environment is selected.

The tree structure also allows hierarchical scenes to be defined. This means that a scene can be completely contained within another scene.

Rationale 85: ***Local referencing***

The scene can be used as a local reference system. Since all known objects have a geometric reference to the scene origin, these can be used to determine the position of the *Robot* in the scene. This local referencing can be used to compensate for global positioning errors. New measurements can also be used to improve the geometric relationships between the objects and the scene.

Rationale 86: ***Efficiency***
By selecting a part of the world model as the active scene, information can be filtered. Much of the data in the world model is only required by other modules if the *Robot* is also in the corresponding scene. Based on the scene concept, the world model can therefore automatically determine which information needs to be sent and which does not.

C.3.5 World Model

Model Name:	Model Kind:	Model Type:	ID:
World Model	VC-M2A	Approach	A11
Addressed Aspects: VC-A2, VC-A2			
Description: The central world representation of AIMM focuses on the physical world. It is extended by virtual objects beyond the purely physical representation. The mission environment is not represented in the world model. An abstract model is chosen for the world model. This is based on an object-based representation. The objects are connected to each other via a tree structure.			
Applies: G14, G15, G16, G17			

Rationale 87: ***Object Types***
The world model consists of objects that are held in a tree structure of geometric relationships. Since an object is an abstract construct, it can represent different aspects of the world. For the AIMM world model, different object types are used that represent different information about the world:

Physical objects represent physical objects. This object type can have properties such as geometric shape, mass, texture, material, etc., which describe the physical properties of a physical object.

Obstacle objects represent obstacle areas in the world. This type of object is used to model a different kind of knowledge about the environment. Due to the incompleteness of information about the world, it is practically impossible and often not useful to model everything as physical objects. However, it is possible to obtain information about the world, e.g. by using sensor systems that provide

depth information. Obstacle objects thus abstract the real world by dividing a spatial area into obstructed and free parts. However, as this information can be subject to errors, voxel spaces are used for the obstacle objects. This form of representation makes it possible to store the reliability of the information or to model the fact that no information is known about certain areas.
The separation of the space by obstacle objects into free, unknown and occupied areas can also be used to introduce further knowledge into the world model. For example, a work table is designed to have objects placed on it. It is therefore to be expected that there will be obstacles on it. The table itself can be integrated into the world model as a physical object. However, the work surface, on which various objects are expected to be in different positions, cannot be represented. More knowledge can be integrated into the world model by using an obstacle object that identifies the work surface of the table as an unknown area.

Shape objects are even more abstract objects, as they exclusively define a geometric shape. These shape objects can have primitive shapes such as cuboids, cylinders or spheres, but also complex shapes. The object type can be used for various purposes, e.g. to define a search volume for an object recognition module, to model safety zones or to define target regions for physical objects. Complex shapes can also be used as sub-objects of physical objects, especially if the physical object is modeled with different shapes.

Transformation objects represent positions and orientations. Since the pose is the relationship between objects and is not stored in the object itself, the object type transformation is the simplest possible object type. The object itself does not contain any information, but exists to implement the relationships.

Sematic objects encode special relationships between objects. As mentioned in the previous sections, the tree structure is limited when modeling object relationships in the real world. Semantic objects are a way to extend the modeling capabilities of the world model. As described in Subsection C.3.3, they can classify objects or create new relationships between objects. For example, semantic objects can be used to model the assembly of multiple parts.

Robot objects represent the *Robot* in the world model. These objects are primarily used to model the relationships between the objects in the world and the *Robot*. Detailed modeling of the *Robot* itself is omitted at this point. The AIMM system is therefore reduced to two robot objects. The robot base represents the platform and all components directly connected to it. Another robot object, the robot flange, is used as a child of the robot base. The transformations between the base and the flange are continuously transferred to the world model and

updated. The details of the *Robot*, such as the individual joints or the status of the pan-tilt unit, are not represented in the world model.
All world objects that relate to the *Robot* base, e.g. objects that are placed on the platform, can be attached as children of the robot base. Objects that are grasped and thus have a strong reference to the flange of the *Robot* are modeled as children of the flange.
This minimalist model of the *Robot* is sufficient to represent the relationship of the system to the objects in the world.

Grasp objects is a special virtual object type for modeling grasps. A grasp has a geometric relationship to its physical object. Therefore, grasps can be added as children of an object. In addition to this geometric relationship, the gripper stores various process parameters such as gripping force and gripping width, approach position and gripping strategies.

Viewpoint objects are objects that model viewpoints. These virtual objects are used to model knowledge about the perception options of the system. For example, several viewpoint objects can be stored as children at a workstation, which together enable reliable perception. Viewpoint objects can of course also be created and stored by the system itself. This means that a viewpoint does not have to be calculated every time a station is approached; instead, prior knowledge can be used.

Approach objects model approach positions. These virtual objects can be used for both the robot base and the robot flange. As a child of an object, they define a relative position for the robot base or the flange in relation to a world model object. These objects can be used, for example, to define the approach positions of the *Robot* at a workstation.

These different object types can be used to store very heterogeneous knowledge in the world model. Many of these object types are virtual objects that do not describe the physical state of the world, but represent knowledge that the *Robot* can use to interact with its environment.

Rationale 88: *Operations on the world model and other changes*
The world model presented in the previous sections represents the world and its state from the *Robot*'s point of view. Since the world is not static and the *Robot* can collect information about its environment, the world model must be changed during execution. There are two main categories of events that lead to an update of the world model: passive and active changes.

Passive changes The first category concerns the integration of new information into the world model. During execution, this information is usually generated by the *Robot* itself with the help of perception modules. However, other modules can also provide new knowledge about the world and make changes necessary:

Object recognition When an object is localized in the scene, there are several possible effects on the world model. The measurement received from object recognition is a relative transformation of the object to the camera coordinates. How this information is integrated into the world model can vary and depends on several factors, such as the reliability of the measurement, the uncertainty of the *Robot* position, the structure of the world model and the context. The decision on how to interpret the measurement is not up to the world model, but the world model must allow for all possible interpretations. The following modifications may be required for object recognition:

- Update relationship (transformation) object - parent object
- Change parent object and create relationship to new parent object
- Update other relationships in the chain e.g. parent to scene
- Add new object to the world model tree
- Remove object from the world model tree

The world model provides these different modification methods. The world model cannot decide which of them should be used.

Scene registration To register the *Robot* in the environment, the transformation between the *Robot* and the scene root must be measured. Since the scene root can be a non-physical object, this registration is usually performed using some landmarks that are physical objects and can be recognized by perception. These landmarks can be specialized markers, but also any other object that can be recognized. The world model can be used to determine the geometric relationship between the landmarks and the scene root to calculate the pose of the *Robot* in the scene. The following modifications may be required for scene registration

- Update (transform) robot - parent object relationship
- Change parent object and create relationship to new parent object

Scene modeling When modeling a new scene, the *Robot* uses its perception modules to generate and update the world model, e.g. to model its own workplace. Depending on prior knowledge, the world model may have a rough structure

of which regions are free, where obstacle regions are located and there may already be some known objects, e.g. from CAD data. The modeling of the scene is essentially a combination of object recognition and scene registration with additional updates of the obstacle objects. However, the updates to the obstacle objects only change the properties of these objects; the relationships and therefore the structure of the world model tree remain unchanged. The following changes may be required for scene modeling:

- Relationship (transform) update parent object
- Change parent object and create relationship to new parent object
- Update other relationships in the chain e.g. parent to scene
- Add new object to the world model tree
- Access to the properties of an object

Viewpoint planning Viewpoint planning refers to the perception modules, but is not a perception module itself. Viewpoints are defined in relation to objects. These objects can be physical objects, but are more often associated with ROIs. The calculation of suitable viewpoints can be computationally intensive and viewpoints can be context-dependent. For example, a window behind the table influences the choice of viewpoints. Therefore, storing the viewpoint in the world model is particularly useful for integrating experience over time. The following modifications may be required for viewpoint planning:

- Update relationship (transform) parent object
- Add new object to the world model tree
- Remove object from the world model tree

Grasp planning Grasps refer to physical objects. Similar to a viewpoint, the grasp can be optimized by the experience of the system. Therefore, new knowledge is generated during the execution of the *Task*, which must be stored in the world model. For example, if the *Robot* has lost the part with the specific grasp, the gripping object can be removed or process parameters such as gripping forces can be changed. The following changes may be required for gripper planning

- Update relationship (transformation) of the parent object
- Add new object to the world model tree
- Remove object from the world model tree
- Access to the properties of an object

These examples represent only some of the possible knowledge-generating events. Due to the simplicity of the structure of the world model, the operations necessary for knowledge integration are bound to these introduced changes. Nevertheless, the structure is complex enough to model different interpretations of measurements.

Active modifications In the first category, the *Robot* was passive and had to integrate new knowledge into the world model, which was either perceived or calculated to keep the world model up to date. Usually there are many ways to interpret this information. The second category is the active category. By definition, the *Robot* must perform a *Task* in the physical world. Therefore, the *Robot* usually has to change its environment. These changes to the environment are intentional, so active operation does not require interpretation of measurements, but can be modeled explicitly.

Move manipulator When the *Robot* move its manipulator the transformation between *Robot* base and flange has to be updated:

- update relationship (transform) object parent object

Move platform When the robots platform is moved the transformation between the scene root and *Robot* base has to be updated:

- update relationship (transform) object parent object

Pick up object When an object is grasped, the parent of the object has to be changed to the *Robot* flange in the world model. The transformation to the *Robot* flange is given by the applied grasp.

- change parent object and create relation to new parent

Place object When an object is placed onto an object in the scene, the parent of the object changes from the *Robot* flange to the object on which the object was placed.

- change parent object and create relation to new parent

C.3.6 World Model Modules

Model Name:	Model Kind:	Model Type:	ID:
World Model Modules	VC-M2A	Approach	A12
Addressed Aspects: VC-A2, VC-A3			
Description: According to G14, the AIMM world model is a central world model. The requirements for this world model are very different from a system perspective. The AIMM world model is therefore based on a modular concept. The information is stored centrally in a database. The interfaces to this data are realized in different modules in order to be able to flexibly cover the heterogeneity of the requirements.			
Applies: G14			

Rationale 89: ***Central Database with Tree Structure***
AIMM uses a central database to store all information. All information is added to this central database. The information is therefore only available in one place. The relations between these data are realized by a tree structure. Each relation contains a transformation between a child and a parent object.

Rationale 90: ***Access to the world model via a modules***
The world model as the central component of the *Robot* contains the knowledge about the world. An equally important point is how this knowledge can be made accessible to the system and how new knowledge can be integrated. Each component needs different information with different requirements, e.g. time constraints, from the world model. Therefore, a module-based concept for accessing the world model was developed, which provides an extensible interface for accessing the central database of the world model. Some general interfaces to this database are presented below:

Basic module The basic module offers the basic functionality for modifying the world model, such as adding and removing objects and changing relationships and properties. Functions for analyzing the scene structure are also available.

File module The file module is used to load and save files that change or save the state of the world model. It is used to define world models and objects in a human-readable file format. The module can be used to integrate prior knowledge, such as the structure of the world, but also the structure of subcomponents, such as physical objects with attached markers and handles.

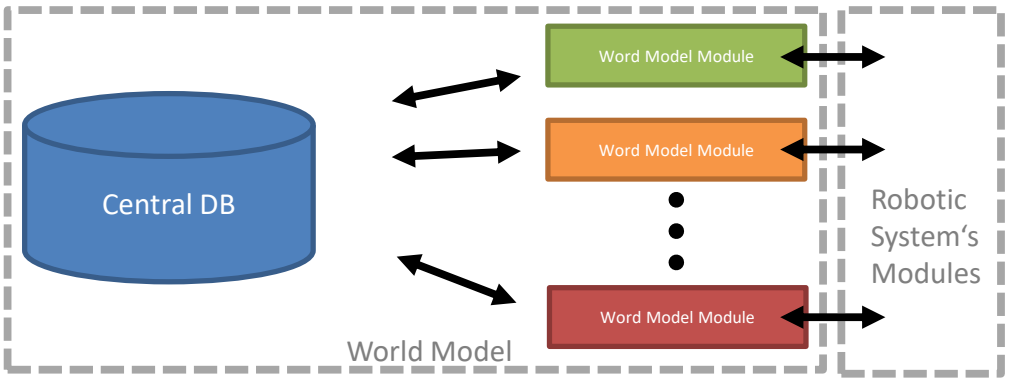

Figure C.6: *Component structure of the world model*

Geometric transformations module Transformations stored in the world model are communicated via this module so that the current relationships between the objects are available for each component.

Object detector module This module extracts the information required by the object detection components from the world model. All object recognition modules require a model of the object. This model often consists of features and the way in which these features are related to each other. Other object recognition modules are based on the shape of the object. The module stores the features and their relationships in the world model and makes them available to the object recognition modules.

Geometric representation module For some components, e.g. motion planners, the geometric representation of the environment is important. This module extracts the geometry of physical objects and obstacles and creates a geometric representation based on the relationships stored in the world model.

Physical module The physical module allows knowledge about physical properties of the world to be extracted. For example, the load data can be queried at the end effector. The physical module calculates the total mass and center of gravity based on the individual masses and the geometric relationships.

The structure of the AIMM world model is shown in Figure C.6. The central component is the database. This is accessed by the *Robot* system via modules. These modules are independent of each other. New interfaces to the system can be created as required. Each module communicates directly with the central database.

C.3.7 Word Model Framework

Model Name:	Model Kind:	Model Type:	ID:
Word Model Framework	VC-M2I	Implementation	I10
Addressed Aspects: VC-A2, VC-A3, VC-A4			
Description: The Neo4j Graph database is used as the central knowledge representation in the AIMM World Model. The various world model modules communicate with this database in order to integrate or extract information. Each world model module consists of two parts. A common part, which is shared by all modules and ensures that the database can only be modified in the specified way. And a specific part that implements the individual operations and interfaces. Each module is started in a separate process.			
Implements: A11, A12			

Rationale 91: ***neo4j as database***
AIMM uses a neo4j database for knowledge representation. With the graph database, the tree structure of the world model can be implemented directly. The nodes of the graph correspond to the objects of the tree, the relations contain the geometric relationship between child and parent object. The decision to use neo4j as a database to store the world state has several reasons:

Persistent data storage and multi-client access The database itself ensures that the transferred information is stored efficiently. neo4j keeps the data persistent, i.e. no information is lost even if the database process is restarted. In addition, neo4j has mechanisms that enable secure simultaneous access to the data from multiple client processes.

Cypher query-language With Cypher, neo4j offers a language with which information can be efficiently extracted based on the graph structure. For this purpose, patterns can be defined that are identified by the database. The AIMM world model uses this interface to extract information based on the tree structure.

NeoModel Library This library implements an object-graph mapping (OGM). It therefore provides a python object that represents the current state of the DB. Information can be extracted and modified here. The AIMM world model uses this interface to access the properties of the objects.

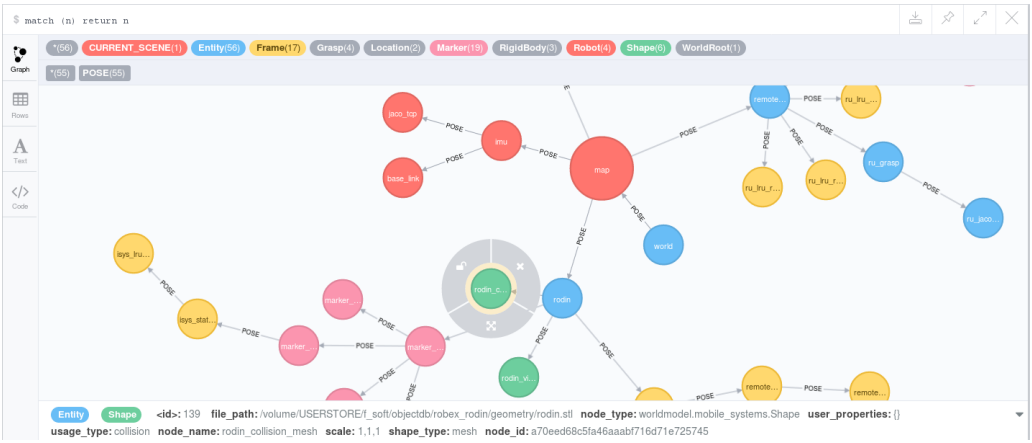

Figure C.7: *Visualization of the world model by the neo4j viewer*

Visualization neo4j offers a web-based visualization of the database, see C.7. This visualization can be used to display the current status, but the database can also be modified manually via cypher queries.

Open Source neo4j is an open source project. If functionalities are missing in the future, neo4j can be extended.

Rationale 92: *World Model Core*

The world model core is based on the NeoModel library and consists of:

core.py Defines the base classes of the world model (objects, geometric relations). The integrity checks for compliance with the tree structure are also implemented here.

operations.py Implements the basic functions for creating, reassigning and deleting objects and making simple queries.

mobile_systems.py Defines the object types that can be used in the world model: e.g. Grasp, Marker etc.

All world model modules use the world model core. This ensures that the additional restrictions, only one parent object, no ring closures, are adhered to so that a tree structure is maintained at all times.

C.4 AIMM Capability Communication

C.4.1 Combining Middlewares

Model Name:	Model Kind:	Model Type:	ID:
Combining Middlewares	VC-M3G	Guideline	G18
Addressed Aspects: VC-A3,VC-A4			
Description: Communication within a *Robot* system is very heterogeneous. However, it has been shown that standardized communication frameworks, so-called middlewares, offer many advantages for a *Robot*. However, there is currently no middleware that can fulfill all requirements on its own. The concept is therefore to combine different middlewares in order to be able to use an optimal solution depending on the requirements.			

Rationale 93: ***No middleware meets all requirements***
Communication in a *Robot* is very heterogonous. There are very different types of data with very different requirements. Middlewares homogenize the communication. This simplifies the modularization of the software and the interchangeability of components. However, there is still no middleware that meets all requirements.

Rationale 94: ***Requirements for the middleware***

Timing The communication of some data is time-critical. The stability of controllers requires that the data is received at a specified rate. If delays occur, this can lead to dangerous system behavior. In addition, a high data rate is desirable for controllers, which further increases the timing requirements.

Dependability Certain data must not be lost. For example, the emergency stop signal is a datum that must be transmitted reliably. The exact timing is less critical than the guaranteed reception of the signal.

Distributed system On a *Robot* the software components run on different computers. Nevertheless, these components must communicate with each other. With a high degree of modularization, it can also happen that modules are started on different computers depending on the workload of the system. The middleware must ensure that communication still functions reliably.

	Timing	Safety	Distributed	Bandwidth	Dynamic Data	Tools	Integration
Sunrise	+	+	-	?	-	-	-
Sensornet	+	o	o	+	-	o	+
ROS	-	-	+	-	+	+	o

Table C.1: *Performance of middlewares used on AIMM*

High bandwidth Cameras in particular generate very large amounts of data that need to be processed by the communication system. Due to a strong modularization of the perception pipeline, the required bandwidth multiplies with the number of processing steps. This quickly results in data volumes that can no longer be fully transmitted via the network.

Flexible data size Some data flows have flexible sizes. For example, the path length of a motion planner depends on the specific task and environment. Middleware must therefore also be able to transfer data with dynamic sizes.

Tools, third-party software and documentation In addition to the technical requirements, a factor that should not be underestimated for middleware is which tools are available for operation, maintenance and updating. How much software with a connection to a middleware exists and how it is documented.

Integration effort Another factor is the integration effort. How high is the hurdle to connect an existing or new software component with middleware?

As shown in table C.1, the various middlewares used on AIMM have strengths and weaknesses. None of the middlewares can solve all requirements satisfactorily, which is why a combination of middlewares is used on the system.

C.4.2 Communication Map

Model Name:	Model Kind:	Model Type:	ID:
Communication Map	VC-M3A	Approach	A13
Addressed Aspects: VC-A1, VC-A3			
Description: In order to be able to combine different middlewares, the communication of the system is divided into different areas. Within these areas, the requirements should be as homogeneous as possible so that they can be covered by one middleware. For AIMM, a sensor area, a coordination area and a real-time area are identified. The coordination area has interfaces to the other two communication areas.			
Applies: G18			

Rationale 95: ***Segmentation of communication into different areas***
Every area transition consumes resources. The aim of subdivision is therefore to have as few transitions as possible. This can be achieved by having few areas but also few transitions between the areas. A homogeneous requirement profile must be identifiable for each individual area.

Rationale 96: ***Sensor area***
The sensor area must be able to process large amounts of data efficiently. It must also enable the implementation of data pipelines.

Rationale 97: ***Control area***
The control area places high demands on the timing of communication. Control loops are closed here. The communication frequency requirements are high, but the bandwidth plays a subordinate role.

Rationale 98: ***Coordination area***
The coordination area has the requirement that many components can be easily connected. To achieve this, the corresponding middleware must work well with distributed systems. The tools are also most important in this area, as this is where the complexity is highest.

Rationale 99: ***Coordination area as a bridge***
As shown in C.8, the coordination area serves as a bridge between the two other areas. It is used to close the sense-act loop at a higher level.

Figure C.8: *Communication map of AIMM*

Rationale 100: ***No subdivision of the software***
The communication map divides the communication of the system into different areas. However, this does not mean that the entire software is divided into different areas. A software module can be in several communication areas at the same time. For example, an object detection module receives the images in the sensor area but is triggered via the coordination area.

C.4.3 Middleware Selection

Model Name:	Model Kind:	Model Type:	ID:
Middleware Selection	VC-M3I	Implementation	I11
Addressed Aspects: VC-A3			
Description: The decisions for the middleware used on AIMM are documented in this model. SensorNet, an in-house development of the institute, is used for the sensor area. The KUKA Sunrise middleware developed by KUKA is used for the control area. Communication in the coordination area is realized with ROS.			
Implements: A13			

Rationale 101: ***ROS as coordination middleware***
For AIMM, ROS was selected as the central middleware for the coordination area. The weaknesses of the middleware in the area of real-time communication and handling large amounts of data can be compensated for by coupling it with the other middlewares. Especially for the high-level areas of the system, such as sequence

control, the weaknesses are of little consequence. This means that the strengths of ROS, which lie above all in its wide range of tools, can be fully exploited.

Rationale 102: ***Cameras via Sensornet***
The cameras are connected via Sensornet. The middleware can handle the large amounts of data from the cameras well thanks to the shared memory approach. In addition, many tools and drivers are available for the cameras. The perception pipelines are therefore implemented with Sensornet.

Rationale 103: ***Control by KUKA Sunrise***
The mobile platform including the manipulator is a KUKA product. Part of this product is also the Sunrise middleware, which enables access to the system. The hardware interfaces, the controllers and also the navigation software are integrated in this framework. Sunrise thus serves as middleware for the control area on AIMM.

C.5 AIMM Capability Computation

C.5.1 Distributed Dedicated System

Model Name:	Model Kind:	Model Type:	ID:
Distributed Dedicated System	VC-M4G	Guideline	G19
Addressed Aspects: VC-A3, VC-A4			
Description: The distribution of processes to different computers in a *Robot* system brings many advantages in terms of the flexibility, reliability and performance of the overall system. In addition, the tasks are not distributed evenly across the various computer units, but individual computer units are specialized for certain tasks. This allows the different requirements of the processes to be taken into account.			

Rationale 104: ***Easy expandability and adaptability***
In a distributed system, individual components can be replaced or expanded more easily. The impact on the overall system is much less when replacing a component than when replacing a central computer. Expansion is also easier, as the infrastructure for connecting several computers is already in place.

Rationale 105: ***Separation of time-critical and computing-intensive processes***
Different process types place different demands on the computing system. Time-critical processes such as controllers must adhere to the required control cycle. A computing-intensive process such as a path planner, for example, requires dynamic access to the memory and maximum utilization of the computing cores. By separating these process types, it is possible to create the best possible environmental conditions in each case without having negative side effects on the other process types.

Rationale 106: ***Reduction of data flows***
Robots with sensors generate large amounts of data that need to be processed. Information often passes through a series of processing steps. By bundling processes on one computer that are involved in a common pipeline, the amount of data between the different computers can be significantly reduced.

Rationale 107: ***Automatic prioritization***
By distributing tasks to different computers, it can be ensured that sufficient resources are available for critical processes. Conversely, the system can be fully utilized on other computers that are intended for computing-intensive tasks.

Rationale 108: ***Higher complexity***
A distributed computer system is more complex than a single central computer. The complexity increases further if different computer types and different connections are used. The concept of a distributed system therefore leads to increased maintenance and operating costs.

C.5.2 Fine Granularity

Model Name:	Model Kind:	Model Type:	ID:
Fine Granularity	VC-M4G	Guideline	G20
Addressed Aspects: VC-A4			
Description: Fine-grained modularization has many advantages in robotics. It simplifies the interchangeability of components. Increases the monitorability of the system. It avoids duplicate implementation and enables greater flexibility. The aim is therefore to achieve a separation into the smallest possible functional units.			

Rationale 109: ***Exchange of subcomponents***
Thanks to the fine granularity of modularization, even small components can be easily replaced. Ideally, the functionality is encapsulated in a process. To replace the functionality, only a new process with the same interfaces needs to be implemented. With coarser granularity, on the other hand, the new functionality must be integrated into an existing process, which usually entails significant restrictions, e.g. in the choice of programming language. In particular, contributions from the community or partners can only be integrated with a lot of effort.

Rationale 110: ***Monitorability of the system***
Communication between the processes is realized by the middlewares used. These usually offer the option of monitoring the flow of information. With fine granularity, the system can be better monitored without additional effort. Errors can be localized more precisely. The performance of individual sub-steps can also be monitored directly via the middleware. With coarse modularization, on the other hand, each process must provide this information individually.

Rationale 111: ***Avoidance of duplicate implementations***
Due to the fine granularity, even small functionalities can be easily integrated. This avoids having to re-implement the same functionalities in different modules.

Rationale 112: ***Alternative solution methods***
Modularization makes it possible to apply different methods to the same problem. One reason for this is that the modules can be easily exchanged. In robotics, however, it is often not possible to identify the absolute best method; methods have different strengths and weaknesses depending on the conditions. By linking the modules via the middleware, different approaches can also be supplied with the necessary information. This parallelization of the methods means that the best solution can always be accessed.

Rationale 113: ***Limits of granularization***
The main factor that determines the limits of granularity is the communication overhead. If the functionality is so small that the communication effort is very large in relation to the method, other approaches, such as the use of shared libraries, must be chosen.

C.5.3 Central Module Management

Model Name:	Model Kind:	Model Type:	ID:
Central Module Management	VC-M4G	Guideline	G21
Addressed Aspects: VC-A4			
Description: Due to the distributed concept and the fine granularity, a large number of modules must be controlled and monitored. For very complex systems, it makes sense to use a higher-level management system. This can take over the coordination of the modules independently of the processes themselves. This software can start the modules and monitor their status. Processes can also be stopped or automatic restarts initiated via this instance.			

Rationale 114: ***Simple integration***
It is important that processes can be easily integrated. Even processes that cannot be changed must be able to be controlled and ideally monitored by the management.

Rationale 115: ***Centralized vs decentralized***
One of the manager's tasks is to provide an overview of the various processes on the individual machines. The visualization should therefore be centralized, which is why it makes sense to also design the manager itself centrally. It would also be conceivable to have one process manager per machine that is connected to a central visualization system. A completely decentralized solution would place higher demands on the implementation of the individual processes.

Rationale 116: ***Dynamic activation of the manager***
The manager starts and monitors the processes. However, the processes must not be dependent on the manager. Once started, they must run independently. The manager must be able to access the processes again at any time and take over monitoring and control.

C.5.4 Process Dependencies

Model Name:	Model Kind:	Model Type:	ID:
Process Dependencies	VC-M4G	Guideline	G22
Addressed Aspects: VC-A4			
Description: Many processes influence each other. An important point is to explicitly record and take into account these process dependencies. The most common process dependencies lead to a prescribed start sequence, as another process is absolutely necessary.			

Rationale 117: ***Types of dependencies***
Dependencies between processes can have various causes. Some types of dependencies are briefly described here:

Data The process can only work correctly if the input data is available. This can of course be handled within the process, but there are implementations where the process goes into an error state for security reasons if the required data is not available.

Infrastructure The process is dependent on a specific infrastructure. For example, ROS processes that communicate with topics must register with the ROS core. If this is not running, the ROS node cannot function properly either.

Initialization Some processes require information from other processes during the start phase in order to initialize correctly.

Rationale 118: ***Dependencies of process states***
Dependencies between processes usually require an operating state of the other process. If, for example, the controller process depends on the joint sensor process, it is not enough for the joint sensor process to have started; it must also supply data.

C.5.5 Individual Runtime Environment

Model Name:	Model Kind:	Model Type:	ID:
Individual Runtime Environment	VC-M4G	Guideline	G23
Addressed Aspects: VC-A4			
Description: Most processes require resources that they receive from their runtime environment. These can be parameters, but also references to program libraries. The large number of processes results in a very large number of parameters that are made available via the runtime environment. Conflicts can also arise if different processes require the same parameter with different values. The concept of the Individual Runtime Environment therefore follows the approach of starting each of the processes in its own runtime environment that is as minimal as possible.			

Rationale 119: ***Versions of software***
A common conflict in large environments is that different processes require the same software in different versions. As the environment variables are usually identical, the correct parameter can only be set for one process.

Rationale 120: ***Trouble shooting***
The more processes are to be started from the same environment, the more extensive the runtime environment becomes. In *Robot* systems with typically more than 100 different processes, it is very difficult to maintain an overview. Errors caused by incorrect or unset parameters are therefore easily overlooked. With a minimal runtime environment, this problem is much easier to manage.

Rationale 121: ***Flexible environments***
Robot in research are often subject to continuous development. New versions of components are regularly integrated or completely new parts are added. With a large shared runtime environment, all effects on the other components must be taken into account immediately. Splitting the runtime environment enables component-by-component testing and integration of new libraries.

C.5.6 Host Types

Model Name:	Model Kind:	Model Type:	ID:
Host Types	VC-M4A	Approach	A14
Addressed Aspects: VC-A4			
Description: Various host types with different requirements are used in the AIMM system. The server host type is used for the administration of central services, such as the shared file system. The sensor host type is used to connect the system's sensors. The Computation CPU host type is used for computation-intensive software. The Computation GPU host type is used for software that runs on GPUs. The real-time host type is used for applications with real-time requirements, e.g. controllers. Each process is assigned to exactly one of these host types.			
Applies: G19, G20			

Rationale 122: ***Correspondency to physical hosts***
The host type defines various requirements for a host. This is also associated with implicit requirements for the physical machine. For example, a Computation GPU host must be equipped with a graphics card. However, it is also possible to assign several hosts to one physical machine. This is particularly the case for systems with a small number of computers.

C.5.7 Process Manager

Model Name:	Model Kind:	Model Type:	ID:
Process Manager	VC-M4A	Approach	A15
Addressed Aspects: VC-A4			
Description: A central software is used as the process manager for AIMM. This means that the process manager generates an individual runtime environment for each process. In addition, the process manager starts and monitors all processes distributed on the various hosts on the *Robot* system and ensures that the dependencies between the processes are taken into account.			
Applies: G19, G20,G21, G22, G23			

Rationale 123: ***Process groups***
To organize the multitude of processes, the process manager offers the option of grouping processes together. A process can belong to several groups. This allows processes to be organized in different ways. For example, there can be a group that contains all processes that are required for execution. However, groups can also organize processes thematically, e.g. all image processing processes are combined in one group. Another possible application is, for example, to group together all processes of a middleware or to group all processes on a host. Groups can also be structured hierarchically, i.e. one group can be part of another group. In this way, it is possible to keep track of hundreds of processes. In addition to pure organization, groups can also be used to control processes. For example, all processes in a group can be started.

Individual Runtime Environment In order to have an individual runtime environment for each process, the process manager offers various mechanisms. In the simplest case, the desired environment variables are defined in the process manager and set in the corresponding runtime environment when the process is started. Another way to set the runtime environment of the process is to pass environment variables from the runtime environment of the process manager to the processes. A third option is to use environment variables from templates. For example, all Ros processes require an environment variable ROS_MASTER_URI. This template can be created centrally and then applied to all ROS processes.

Monitoring In order to monitor the processes as generically as possible, the process is not only monitored as a process, but the console output of the process is also used to obtain information about the process status. For example, most processes go through an initialization phase in which they are usually not yet able to perform their task. The process manager can recognize this state based on the console output. Another use case for text-based monitoring is warnings or error messages. These can be recognized with the help of the process manager and can be highlighted centrally.

Dependencies Dependencies between processes are monitored with the process manager. When a process is started, the system checks whether there are dependencies to other processes. If these exist, the relevant processes are started first. The actual process is only started when these are ready for operation. The process manager can also recognize during operation if dependencies are violated. If a process on which another process depends is terminated, this leads to warnings and, depending on the configuration, to automated restarts of the processes involved.

C.5.8 LN Manager

Model Name:	Model Kind:	Model Type:	ID:
LN Manager	VC-M4I	Implementation	I12
Addressed Aspects: VC-A1, VC-A2, VC-A3, VC-A4			
Description: The LN Manager is used for the implementation of AIMM process management, as it can fulfill all the relevant requirements of the AIMM system. The LN Manager is also used for communication parameterization. Furthermore, the manager distributes the processes to the different hosts of the system based on the host type. The software tool is part of the LN middleware [110]			
Implements: A9, A10, A13, A14, A15			

Appendix D

AIMM Skill View

This chapter describes the *Skill View* of the AIMM system. It begins with an overview of all identified *Models* and their relationships with each other. The individual *Architecture Models* are then presented.

D.1 AIMM Skill Overview

***Guideline Models*:**

- Finest Granularity
- State-based Composition
- Strong Hierarchy
- Reuse of skills
- Autonomy by Static Routines
- Explict Formalization
- Open Approach
- Recources Type
- Forward Recovery
- Backward Recovery
- Fault Compensation
- Fault Removal

***Approach Models*:**

- Hierarchical Statemachine with Data Flow
- Skill Primitives
- Skill Libraries
- Resources Manager
- Operations on Resources
- Hierarchical Error Propagation
- Execution Logging

***Implementation Models*:**

- RAFCON Libraries
- RAFCON Statemachine Design
- RAFCON Recource Management
- RAFCON Handling & Logging

D.2 AIMM Skill Type

D.2.1 Autonomy by Static Routines

Model Name:	Model Kind:	Model Type:	ID:
Autonomy by Static Routines	VS-M2G	Guideline	G24
Addressed Aspects: VS-A2			
Description: The core of the AIMM *Skill* concept is to achieve autonomous behavior through static routines. A *Skill* calls many *Subskills* and *Capabilities* in a fixed structure to ensure that the *Task* is successfully solved even if problems arise. This means that the flow structures are deterministic, but this does not mean that the *Robot*'s behavior is deterministic. Based on sensor data or e.g. through *Capabilities* based on heuristic procedures, the system reacts differently to similar *Tasks*. AIMM pursues this approach very intensively by solving even complex *Tasks* using static routines.			

Figure D.1: *The Relations of the Skill View Models*

Rationale 124: ***Explainability of the behavior***
A major advantage over other approaches to achieving autonomy is the explainability of behavior. The behavior cannot be predicted exactly, but after execution it is possible to analyze exactly why the system reacted the way it did in this situation. This is crucial in order to be able to localize and resolve any malfunctions.

Rationale 125: ***Iterative process possible***
The routines can be constantly expanded and improved. It is therefore not a question of finding a method, a model or an algorithm that can solve all conceivable problems. Rather, the routine must be suitable for the problems at hand. If new requirements arise, the routine can be extended accordingly. This allows the complexity of the flow control to be reduced to a necessary level.

Rationale 126: ***Solving complex Tasks with static routines***
Static routines are the appropriate approach for simple *Tasks* that are difficult to describe symbolically. For sequencing problems of *Subtasks*, however, schedulers are the more suitable method. In between, there are *Tasks* that can in principle be solved with both approaches. These are, for example, simple manipulation *Tasks*. There are two main factors in favor of a routine-based approach:
Firstly, a routine can deal better with incomplete knowledge, as it allows actions suitable for a situation to be stored directly. It is not essential to know why these actions are carried out or what they achieve in detail. The less information is available, the more reasonable it is to fall back on general strategies. A range of different approaches, applied according to the principle of trial and error, can also solve such situations.
In a planning approach, however, these concepts must be integrated into the general model of actions, including their effects on the state of the world. Especially when the impact of an action is unclear due to insufficient information, modeling becomes very difficult.

The second factor is *Task* variance. If an identical *Task* has to be performed very often, a suitable routine is often more efficient than a planning-based approach that generates a new solution for each repetition. For AIMM, in an industrial context, usually both factors are relevant since it is a partially unknown environment and repetitive *Tasks*. Therefore, many *Tasks* are solved by routines.

Rationale 127: ***Overhead through routines***
The routines must be kept generic so that they can be used in different situations. This leads to additional overhead, which reduces the efficiency of the system compared to other approaches. This is also due to the fact that with encapsulated routines it is difficult to optimize across several routines. If, for example, several objects are to be picked up, the pick object routine is used several times. This may result in the platform having to be repositioned several times. It would be more efficient to find a common platform position from which all objects can be picked up. Such optimizations are difficult to implement with static routines. At AIMM, nevertheless, this approach is chosen in order to achieve a high degree of autonomy and robustness at the same time.

However, in order to achieve an increase in efficiency, it can be useful to add an optimization step, e.g. Brunner et al. [24].

D.2.2 Finest Skill Granularity

Model Name:	Model Kind:	Model Type:	ID:
Finest Granularity	VS-M2G	Guideline	G25
Addressed Aspects: VS-A2			
Description: By definition, the usage of a *Skill* solves exactly one Task. Skills therefore link the goal of an action with its implementation. However, no direct semantic information is stored within the *Skill*. The individual sub-components, if they are not Skills, only contain the implementation of the execution. Therefore, a finer granularity of the Skills leads to a better link between the declarative representation and the procedural representation. This information is important in order to be able to react to unforeseen problems. For AIMM, the Skills are therefore defined as finely granular as possible.			

Rationale 128: *Skill: Combination of semantics and execution*
Semantics and execution define a *Skill*. If a different execution is selected for the same *Task*, this results in a different *Skill*. Conversely, the same execution can be used for different *Tasks*, which in turn leads to different *Skills*. This combinatorics results in a large number of *Skills*. A fine granularity helps to keep the execution variants manageable, as only a few actions are called within a *Skill*. A *Skill* that only contains one *Capability* has fewer variation options than a *Skill* with many linked *Capabilities*.
This does not change the combinatorial complexity, but helps to maintain an overview. *Skills* with few components are clearer and, if *Subskills* are used, contain further semantic information that helps to organize the *Skills*. This finer granularity of semantics helps to interpret the variation when replacing a *Subskill*, for example. If the new *Subskill* has the same *Task*, the execution has been changed, but if the new *Subskill* has a different *Task*, the strategy has been changed. A fine granularity is also helpful for error detection, as even smaller action sections have a defined *Task*. This *Task* accomplishment can often be checked better than the correct execution itself. Fine-grained semantics therefore also enable more specific error detection.

Rationale 129: *Routines vs. behaviors*
The classic behaviors, e.g. from the subsumption architecture [17], link different *Capabilities* directly via their output values. The *Robot* behavior results from the combination of several *Capabilities*. Certain *Tasks*, such as navigating to a position while avoiding obstacles, could be realized with this approach. For more complex *Tasks*, however, adjusting the weights proved to be impractical. Especially when opposing *Tasks* such as obstacle avoidance and driving

through a bottleneck come together, it is difficult to find a generally valid parameter set. Behaviors lack the semantics that allow the *Robot* to assess the situation. For example, the system has to drive close to an obstacle because there is a bottleneck to pass. This situation-dependent parameterization can be implemented using routines. In AIMM, behaviors are therefore avoided and replaced by routines. Behaviors are only used for less complex *Tasks* that require a fast reaction time.

D.2.3 Skill Primitives

Model Name:	Model Kind:	Model Type:	ID:
Skill Primitives	VS-M1A	Approach	A16
Addressed Aspects: VS-A1			
Description: *Skill Primitives* are the most detailed structure of *Skills*. A *Skill Primitive* contains only one *Capability* and optionally *D&I* components for interpreting the *Capability*'s result.			
Applies: G24, G25			

Rationale 130: *Minimal Skill*
The finest granularity of a *Skill* is achieved when the *Skill* only contains a single *Capability*. These so-called *Skill Primitive* Subsection 4.6.3 essentially have the function of defining the purpose of execution In addition to calling the *Capability*, most *Skill Primitives* contain components that interpret the return value of the *Capability* to determine the result of the *Skill*.

Rationale 131: *Use of low level semantics*
The *Capability* itself contains no information about why it is used. Thus, the same *Capability* can be executed for very different reasons. For example, a relative, cartesian movement in the impedance controller can be used to bring the end effector to a relative position. However, the same *Capability* can also be used to haptically reference to an object. Only the purpose of the execution allows the result to be analyzed. In the first case, the "movement" *Skill*, the execution is successful when the goal position is reached. In the second case, the "referencing" *Skill*, the execution is only successful if the goal position is not reached, but a force is created instead. Storing the semantics therefore already enables qualified error detection at this level. For this reason, AIMM attempts to encapsulate every *Capability* call in a *Skill*.

D.3 AIMM Skill Hierarchy

D.3.1 Reuse of skills

Model Name:	Model Kind:	Model Type:	ID:
Reuse of skills	VS-M1G	Guideline	G26
Addressed Aspects: VS-A1			
Description: The more complex *Skills* are kept as generic as possible to enable reuse. Therefore, the *Skills* are not optimized for a specific situation, but an attempt is made to create the widest possible range of applications. To achieve this, the *Skills* also contain components that generalize their applicability and reduce dependencies on the system and environment status. In addition to the generalization, the fine granularity also contributes to the reusability of the *Skills*, as each *Subtask* is addressed in a *Subskill* that can be used elsewhere.			

Rationale 132: ***Complex Skills***
High level *Skills* can become very complex and contain several hundred *Subskills* and *Capability* calls. Due to the structure of high-level *Skills* as routines, often only a fraction of the components they contain are activated during execution. This makes the design and, above all, the testing of these *Skills* very complex. The aim is therefore to use as few, but well-tested *Skills* as possible, which in turn are based on relatively few, well-tested and reusable *Subskills*.

Rationale 133: ***Reduction of dependencies***
High-level *Skills* should have as few dependencies as possible, as this allows them to be used more flexibly. This can be achieved by actively checking the influenceable dependencies within the *Skill* and if necessary creating them. For example, a "pick object" *Skill* can have the accessibility of an object as a dependency. Alternatively, the "pick object" *Skill* can also contain a reachability check and a routine that ensures reachability if necessary.
This approach increases the complexity of the *Skill*, but allows it to be used in many situations. This increases reusability.

Rationale 134: ***Incremental design***
The incremental design of *Skills* continuously improves the abilities of the *Robot*. By reusing *Skills*, routines that are not currently being worked on also benefit from improvements elsewhere.

D.3.2 Strong Hierarchy

Model Name:	Model Kind:	Model Type:	ID:
Strong Hierarchy	VS-M1G	Guideline	G27
Addressed Aspects: VS-A1			
Description: Each *Skill* has the purpose of fulfilling a *Task*. The *Skill* annotates the components it contains with the objective of the *Task*. However, the semantics of the individual components are not specified in detail. AIMM therefore follows the concept of combining the components into *Subskills* as soon as they jointly fulfill a *Task* that goes beyond the *Tasks* of the individual components. This concept leads to a very strong hierarchization of *Skills*, but without introducing rigid layers.			

Rationale 135: *No static hierarchy levels*
The concept of encapsulating every identifiable *Task* within a *Skill* in a *Subskill* results in a large number of hierarchy levels. However, these are not rigidly defined, but result from the identifiable *Tasks* and *Subtasks*. There is therefore no fixed categorization of *Skills* into hierarchy levels. Each *Skill* can be used as a *Subskill* in another *Skill*. This results in a dynamic number of hierarchy levels. The same *Skill* can also be used as a *Subskill* in different *Skills* at different levels.

Rationale 136: *Hierarchization is parallelization of Tasks*
Hierarchization automatically results in parallelization of the *Tasks*. Each hierarchy level adds a *Task* that must be completed simultaneously. In contrast to a flat plan, in which only actions are sequenced, the hierarchical *Skill* approach creates a context that allows conclusions to be drawn about the system status.

Rationale 137: *Hierarchy structures and limits*
The strong hierarchy keeps *Skills* with thousands of *Subskills* manageable. The resulting structure is very helpful for human developers both when programming and when testing and monitoring execution.
However, the hierarchy also entails various restrictions. For example, processes are strictly divided into their *Tasks*, which are then executed sequentially. For technical reasons, however, it would often make sense to start with parts of the next step before the previous one has been completed. Hierarchization does not allow this directly. Another disadvantage of the hierarchy is that it only allows one subdivision. However, different types of *Subtasks* can often be identified, some of which overlap. The tree structure provided by the hierarchy does not allow this to be modeled.

D.3.3 Skill Libraries

Model Name:	Model Kind:	Model Type:	ID:
Skill Libraries	VS-M2A	Approach	A17
Addressed Aspects: VS-A1, VS-A2			
Description: To increase the reusability of *Skills*, AIMM uses the *Skill* library approach. A *Skill* library can be created from any *Subskill*. When a *Subskill* is converted into a library, this part becomes independent of the parent *Skill*. This makes it possible to use this library as a *Subskill* anywhere else. Changes to the library can only be made directly in the library. These then affect all instances of the library. Local, application-specific adjustments to the library are no longer possible.			
Applies: G24, G25, G26, G27			

Rationale 138: ***General structure of Skill libraries***
A library is a state machine of any complexity that can be integrated into other *Skills* as a *Subskill*. For this purpose, the state machine is encapsulated in a so-called library state. Like all states, a library state also has a logical input and at least one logical result. In addition to the logical connection, a library can also define incoming and outgoing data flows. Integrating a library therefore works in the same way as creating a subordinate hierarchy level. Conversely, the use of a library in the higher-level state machine creates at least one new hierarchy level.

Rationale 139: ***Skill library vs. Skill templates***
A library is integrated like a hierarchy state. However, the subordinate levels are defined in the library and cannot be changed from the superordinate state machine. Changes in the library in turn affect all instances of the library. This has the advantage that all improvements to a *Skill* resulting from the iterative development process are used automatically. Due to this implementation, it is important that the libraries are implemented as generically as possible in order to be able to use a wide range of applications.
The disadvantage of central libraries is that specific adaptations must always be compatible with all other applications of the library. A *Skill* template that can be copied and then changed locally would solve this problem. Local changes would then not affect the library and the other instances. However, it is then very time-consuming to transfer general improvements to all instances. With AIMM, the advantage of the iterative development process prevails, which is why *Skill* libraries are used.

Rationale 140: ***Hierarchical Libraries***
A library can in turn contain any number of libraries. These are integrated into the library as library states and thus automatically into a lower hierarchy level. A library can also contain the same library multiple times on different hierarchy levels. The only restriction is that a library cannot contain itself.

Rationale 141: ***Fine granularity***
The granularity of the libraries should also be kept as low as possible. This means that as soon as several states are linked together and this combination is required at another location, a library should be created from these two states. This can then be used in both places, thus avoiding code duplication.

D.3.4 RAFCON Libraries

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Libraries	VS-M3I	Implementation	I13
Addressed Aspects: VS-A1, VS-A2			
Description: RAFCON offers a library concept. Any existing state machine can simply be converted into a library via the GUI. This library is then made accessible to other state machine implementations via various concepts. However, online modification is then no longer directly possible. Nevertheless, RAFCON offers the option of opening several state machines simultaneously. The library can then be modified in a separate window, but synchronization at runtime does not take place. RAFCON libraries are therefore used on AIMM to implement *Skill* libraries.			
Implements: A16, A17			

D.4 AIMM Skill Composition

D.4.1 State based Composition

Model Name:	Model Kind:	Model Type:	ID:
State based Composition	VS-M3G	Guideline	G28
Addressed Aspects: VS-A3			
Description: Robots and their flow control are complex, as many components and states of the system and its environment must be taken into account. A state-based composition supports the developer by explicitly specifying a part of the system state. When programming, it is not necessary to analyze the history of past commands to determine the current state, but the position in the flow itself represents this.			

Rationale 142: ***State of the flow***
The *Robot*'s behavior must adapt to the situation. The appropriate response depends on the *Task*, the system state and the environmental conditions. However, the state-oriented composition does not refer to this overall state, but to the small section that contains the state of the flow. So essentially, what are the current *Tasks* to be solved. This focuses the programming on a specific situation of the flow. This approach does not solve the problem that the majority of the state is only very incompletely known at the time of programming.

Rationale 143: ***Local programming***
State-oriented programming makes it possible to react to a specific situation. Possible errors or limitations of the flow can be solved for specific problems without being a universal solution. For example, the reaction of a *Robot* to unexpected contact with its environment must be very different depending on the situation. If the *Robot* is in a human-robot collaboration, it is likely that an emergency stop will be triggered immediately or the *Robot* will have to be switched off. If the contact occurs in a "reach into the box" scenario, it is probably sufficient to move the manipulator back so that the object position estimation can update the environmental situation. The best solution would be to identify the reason for the different behavior and then react generically to the semantic situation (human in the workspace, environment model incorrect, ...). However, this generalization is difficult to achieve in real applications. Local programming offers the possibility of implementing a direct solution. These programs can then be analyzed in a further step to identify patterns.

D.4.2 Hierachical Statemachine with Data Flow

Model Name:	Model Kind:	Model Type:	ID:
Hierachical Statemachine with Data Flow	VS-M3A	Approach	A18
Addressed Aspects: VS-A1, VS-A2, VS-A3			
Description: In order to apply the guidelines G24, G27, G28, a statemachine-based approach to flow control is chosen for AIMM. The logic flow is represented by states that are connected to other states via transitions. To enable a hierarchy, states can in turn contain state machines. To enable further linking of the various execution states, the pure state machine approach is extended by data flows.			
Applies: G25, G24, G27, G28			

Rationale 144: ***flow control state <> system state***
The hierarchical state machine used as flow control represents the state of the flow and therefore only a very small part of the system state. This is a fundamental difference to state machines that represent the states of a classic automaton, for example.

This is illustrated in Figure D.2. In the classic state machine, the actions are the transitions, as these change the state. In flow control, on the other hand, the states represent the actions. The system states, in turn, are the transitions between the flow states. This approach has the advantage that not all system states have to be modeled, but only the actions that a *Robot* can execute.

Rationale 145: ***Data flow & discrete states***
Modeling the logic flow as a state machine implies that all actions of the *Robot* are modeled as discrete states. At the high hierarchy levels, this encapsulation into states works directly: state 1 executes *Task* 1, state 2 executes *Task* 2, etc. On the lower hierarchy levels, however, capabilities must be called. These usually require call parameters and often return values. In order to encapsulate actions strictly in states, an infinite number of states would have to be created that only encapsulate the parameters. For example, move TCP 1 cm in the z-direction, move Tcp 2 cm in the z-direction, etc. This is not practicable, but as this fine granularity is required for the application of G25, the state machines are extended by data flows. The logical flow is modeled by the transitions between the states. However, parameters and results are exchanged via data flows that are independent of the logical flow.

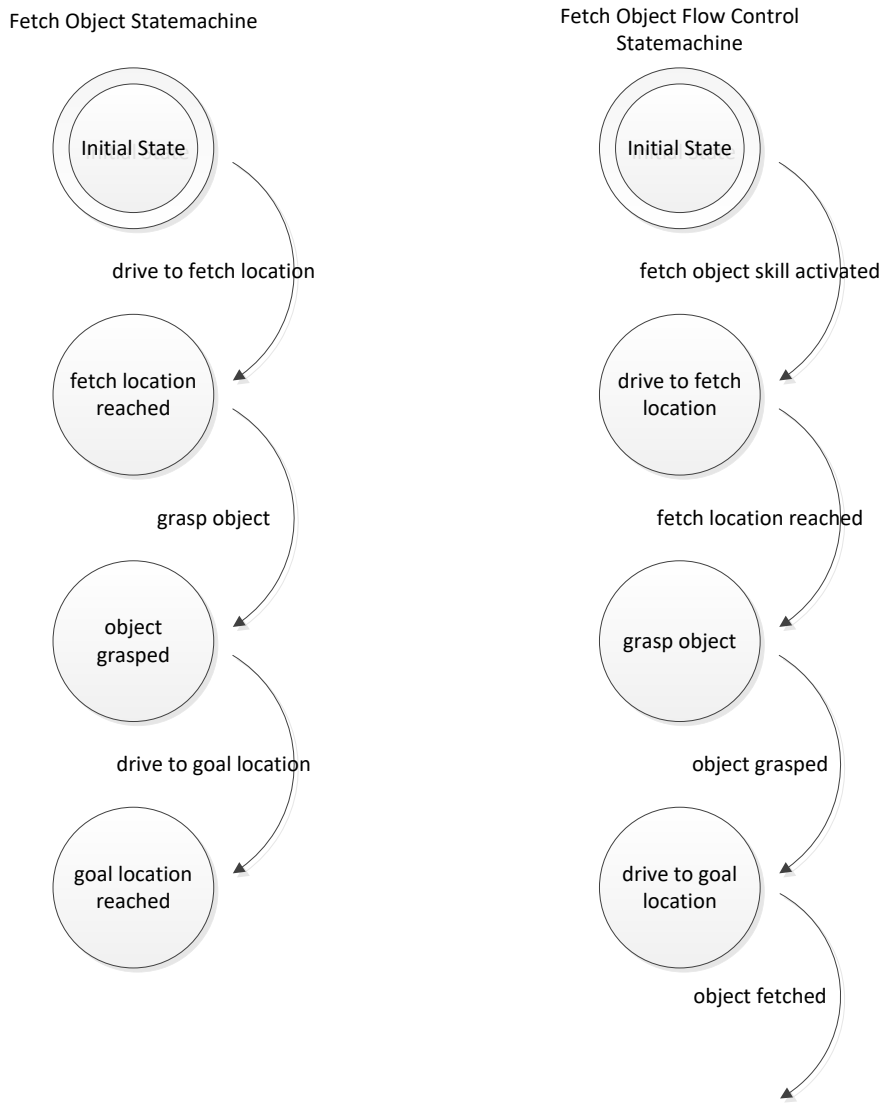

Figure D.2: *Example of a fetch object Task implemented as statemachine (left) and as flow control statemachine (right)*

D.4.3 RAFCON Statemachine Design

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Statemachine Design	VS-M3I	Implementation	I14
Addressed Aspects: VS-A3			
Description: RAFCON offers a graphical user interface for programming and executing complex hierarchical state machines. The state machine can also be evolved and modified during execution. This combination enables very efficient iterative development of state machines. Data streams can be used both to link states and to parameterize them. RAFCON is therefore used for AIMM for the implementation of A18. Skills of AIMM are therefore implemented as RAFCON statemachines.			
Implements: A18			

D.5 AIMM Skill Resources

D.5.1 Explicit Formalization

Model Name:	Model Kind:	Model Type:	ID:
Explicit Formalization	VS-M4G	Guideline	G29
Addressed Aspects: VS-A4			
Description: The *Skills* contain many dependencies that need to be taken into account and have effects that need to be considered. Only a small part is explicitly formulated via interfaces, logic and data as well as semantic annotation of the *Task*. All other effects, such as unintended effects, preconditions, etc., must nevertheless be taken into account. Generic error handling or automatic generation of new sequences is therefore impossible, as this implicit information is missing. The concept of explicit formalization in *Skill* resources therefore requires all dependencies and effects of a *Skill* to be modeled explicitly. This includes all external influences on the *Skill* as well as all effects caused by the *Skill*.			

The *Skill* concept of AIMM enables a robotics expert to develop routines on the basis of which the *Robot* can autonomously solve *Tasks* in an environment that is only partially known. The routines contain information on which *Skills* are linked and how, and which data flows occur.

Depending on the maturity of the routines, the *Task* and the environment, this is sufficient to define the *Robot*'s behavior.

However, if the conditions change, e.g. due to a change in the environment or a new *Task*, and a routine fails, it is often difficult to find the exact cause of the error. Error analysis and behavioral adaptation in turn require a lot of expert knowledge and the system itself can do little to help.

The basic problem is that the necessary information is only implicitly coded in the routines, if at all. A robotics expert can only solve this problem by interpreting the descriptions of the *Skills* and using background knowledge. However, this is difficult to do with larger routines, so that changes to routines often create new problems.

In Subsection 4.6.2 this level was introduced as a resource level. A central concept of AIMM is to make this information explicit. In the AIMM system, all dependencies and effects of a *Skill* are modeled as *Skill* resources.

Rationale 146: ***Automatic error handling***
Skill resources are an important tool for the development of automatic error handling strategies. Many error cases in robotics are not exclusively due to the currently executed action, but are the result of error chains. These are often difficult to trace. There are often several possible causes of error for one and the same problem. It is also often not trivial to determine the exact point at which an error occurs, as methods work with tolerances. The question of whether one method is too imprecise or another method is too sensitive for the requirements often cannot be answered clearly.
Skill resources help to reduce this uncertainty by explicitly defining the prerequisites as well as the results. If a prerequisite for a *Skill* is missing, a propagated error is not due to this *Skill*, but must be looked for elsewhere.
Skill resources therefore help to find the source of the error and identify the type of error. Based on these results, suitable error handling strategies can then be applied automatically by the system, e.g. to fulfill a previously unfulfilled precondition for a *Skill*.

Rationale 147: ***Optimization of the process***
The *Skills* are developed and tested by robotics experts for AIMM. This implies that the behavior corresponds to a human-compatible implementation. Parallelizations, non-semantic sequencing, etc. are difficult for humans to understand and are therefore rarely used. This results in the generated behavior containing restrictions that do not result from the *Task* to be solved, but from the type of generation. By using *Skill* resources, these artificial restrictions can be recognized and resolved. This can significantly increase the efficiency of the *Skill* [24].

Rationale 148: ***Same Skill but different purpose -> different resources***
By definition, a *Skill* solves a *Task*. However, what is not defined in a *Skill* and cannot be defined is why this *Task* is being solved. This purpose can only be defined at a higher level. Depending on the purpose of the *Skill*, there are different effects and dependencies. The *Skill* resources can therefore only be partially assigned to the *Skill*, some resources must be linked to the *Skill* instance, which is realized by the higher-level *Skill*.

D.5.2 Resources Types

Model Name:	Model Kind:	Model Type:	ID:
Resources Types	VS-M4G	Guideline	G30
Addressed Aspects: VS-A4			
Description: The *Skill* resources are very heterogeneous. Different types of resources can therefore be defined. This grouping of resources helps to systematically determine the dependencies of a *Skill*. The following resource types are defined for AIMM: world state, robot state, module state and process state.			

Rationale 149: *World state*
A *Robot* solves *Tasks* in its physical environment. This automatically creates a dependency between the *Skill* instances via this shared environment. Changes made by one *Skill* instance therefore affect the other *Skill* instances.
To make this explicit, it is necessary to define which world state resources are required, consumed or generated for a *Skill* instance.

Rationale 150: *Robot state*
The physical *Robot* is another component that connects the *Skill* instances. All *Skills* run on the same *Robot*, which creates certain dependencies. For this reason, all resources that are assigned to the physical *Robot* are summarized in the *Robot* state resource type.
Robot state resources include all resources that model the state of the system, such as the battery voltage. In addition, resources that model the relationship to the environment, e.g. *Robot* has contact with the environment, are also assigned to this type.

Rationale 151: *Module state*
The AIMM software runs on separate modules. These modules can be in different states. The execution of *Skills* therefore depends not only on the individual *Capability* call with its input data and results, but also on the state of the corresponding module. Module state resources are used to model these dependencies. An example of such a module state resource would be a path planning *Capability*. Starting from the current configuration, a collision-free path to the target configuration is calculated when the *Capability* is called. However, these calculations are based on the environment model stored in the path planning module. For a correct result, the environment model in the planning module must be up-to-date. This dependency is modeled as a module-state resource.

Rationale 152: *Processes state*
A process can be in different states. For example, running, being started, stopped, etc. Functionalities are available depending on these states. The *Skills* therefore depend on the process state, which is modeled by the process state resource type.

D.5.3 Open Approach

Model Name:	Model Kind:	Model Type:	ID:
Open Approach	VS-M4G	Guideline	G31
Addressed Aspects: VS-A4			
Description: It is difficult to create a complete resource model for a complex system. During the development, but also during the use of an autonomous *Robot*, new situations can always arise that reveal incompleteness in the resource model. The concept for AIMM is therefore the extensibility of the resource model. It must be possible to adapt and extend the resource model during development and also at runtime. In addition, it must be possible to map any dependencies as resources. For this reason, a very open approach is pursued in which any dependencies can be mapped as resources.			

Rationale 153: ***Systematic and unambiguous***
The open approach allows any dependencies to be formulated directly as a resource. For example, a resource could be *parameters in the permissible range*. The problem that arises from this is that resources with the same name represent different circumstances, as different parameters are naturally relevant for each *Skill*. This means that the uniqueness of the resource is lost, as the resource exists for one *Skill* but not for another. Furthermore, it is not possible to find a unique type for this ressource. General rules based on the type can therefore not work. This also violates the systematic structure of the resources.

Rationale 154: ***Discovered dependencies must be modeled explicitly***
The resource model enables the explicit modeling of dependencies. However, the developer is not fully aware of all dependencies. Changes elsewhere or new application situations can reveal dependencies that were previously not taken into account. These dependencies can then often be identified during troubleshooting. It is therefore important to also record these subsequently gained insights.

Rationale 155: ***Even imprecisely formulated resources can be helpful***
Ideally, resources should be precisely specified and verifiable. However, if a dependency cannot be identified in this way, unspecific resources can also be useful. For example, an object recognizer requires a good quality image in order to be able to perform correct recognition. The resource "good image" in itself does not say much, but offers a starting point for initiating further steps in the event of frequent errors in a component. For example, another image can be recorded.

D.5.4 Operations on Resources

Model Name:	Model Kind:	Model Type:	ID:
Operations on Resources	VS-M4A	Approach	A19
Addressed Aspects: VS-A4			
Description: The AIMM resource model is an open model. This means that any dependencies can be modeled as resources. The possible operations in the resource model, in contrast, are uniformly defined. Resources can be required, generated, consumed and blocked.			
Applies: G29, G30, G31			

Rationale 156: *Limited set of operations*
The operations on open resources could be arbitrarily complex. However, the operations required for AIMM to model the dependencies between the competencies can be reduced to four operations:

create: *Skills* can create resources. These are available after the *Skill* has been executed.

require: *Skills* can require resources. The *Skill* can only be executed if the corresponding resources are available.

block: *Skills* can block resources. Some resources are required exclusively by *Skills*. During the entire execution of the *Skill*, no other *Skill* can access this resource. The resource is only released again after execution.

consume: *Skills* can consume resources. These resources are required exclusively by *Skills*. The resource is deleted when the *Skill* is executed.

By restricting the operations, the resource model can be efficiently adapted to the execution by the resource manager. Changes to resources, at least by *Skills*, only take place when a *Skill* is started and ended. Due to the strong hierarchy of *Skills* G27 and the fine granularity G25, this is not a problem for the AIMM system.

Rationale 157: *Skill hierarchy*
Due to the hierarchical structure of the *Skills*, several *Skills* are usually active at the same time. Even if they are embedded in each other, they do not share the same resources. This means that if a *Skill* at a higher level blocks a resource, all the *Skills* below it cannot access this resource. For this reason, it is important to block resources at the lowest possible level to prevent subroutines from being blocked.

D.5.5 Resources Manager

Model Name:	Model Kind:	Model Type:	ID:
Resources Manager	VS-M4A	Approach	A20
Addressed Aspects: VS-A4			
Description: A central resource manager is used to manage the *Skill* resources. Resources can be created and managed with this manager. The *Skill* instances can access the resources at runtime via the resource manager. The Resource Manager is also responsible for monitoring the resources. In addition, the resource manager implements general resource rules.			
Applies: G29, G30, G31			

Rationale 158: *Central resource manager*

The resource manager is created as a central module, as many resources model dependencies between components, modules, etc. In addition, the *Skills* are executed in a central master state machine that contains all *Skill* instances. This means that the *Skills* already have a centralized structure. It makes sense to use this structure for modeling the dependencies between the contained *Skill* instances. However, centralizing resource management also has disadvantages:
The bigger the scope of resource management, the higher the risk of violating the uniqueness of the resources. This means that one resource name is used for several different resources. The open approach and the iterative development of *Skills* in particular intensify this problem. Some resource types are also strongly linked to components and modules. For example, all process state resources can of course be linked to processes. This would also make it possible to build decentralized resource administrations that manage the process resources on each host. In this way, certain process resources, such as host utilization, could be monitored and managed more easily and efficiently.

Rationale 159: *Complex resources*

The resources represent the dependencies between the *Skill* instances. For each *Skill* instance, it is explicitly modeled which dependencies must be fulfilled, i.e. which resources are required, but also which dependencies are fulfilled, i.e. which resources are generated. From the perspective of the *Skill* instance, this is clear. However, when many *Skills* interact, the problem arises that the dependencies between the resources are not modeled. This means that the consumption of one resource can have an indirect effect on many other resources, as they are dependent on each other. The consequence of this is that either each *Skill* instance contains a large number of resource effects, as each dependent resource must be entered. This list must then be updated each time a new resource with a dependency is added. This leads to a high level of complexity that is not scalable for systems with many *Skills*. Alternatively, resources could be selected so that they have no dependencies on each other. This would contradict the open approach, as it would no longer be possible to model any dependencies

as resources.
So-called complex resources are therefore introduced in the AIMM. Complex resources have dependencies on other resources. These are managed by the Resource Manager, which monitors all dependencies and implements the respective effects. An example of a complex resource is *the_robot_is_ready_to_drive*. This resource can only exist if *emergency-stop-free* and *battery_ok*. If one of the two dependencies is violated, the resource manager also removes the complex resource.
Conversely, the creation of a *the_robot_is_ready_to_drive* resource also means the creation of the *battery_ok* and *emergency-stop-free* resources. A *or* link is also possible for complex resources. For example, a resource *object_is_visible* can exist if one of the resources *object_in_field-of-view[sensor_1]* or *object_in_field-of-view[sensor_2]* exists.

Rationale 160: *General resource rules*
There are resources that are not modified by the *Skills* but for which there are clear rules. For example, a *Capability* can only be used if the corresponding process is running. There is therefore a general dependency between *Capability* calls and the corresponding process resources. This dependency is generally valid and is checked in the event of an error with the help of the resource manager.

D.5.6 RAFCON Resource Management

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Resource Management	VS-M4I	Implementation	I15
Addressed Aspects: VS-A4			
Description: RAFCON does not offer a built-in resource manager. However, the Global Variable Manager (GVM) from RAFCON is used on AIMM to save the current resource status. For this purpose, simple data structures are created which model the respective resource. RAFCON libraries that can access these values are used to realize the operations. The resource interfaces of the respective *Skills* can be defined in the respective state machines using the Semantic Data plugin.			
Implements: A20, A19			

D.6 AIMM Skill Dependability

Robots work in a real, physical environment. Uncertainties, modeling errors, sensor noise and many other effects result in errors that must be taken into account so that the *Robot* can perform its *Task* reliably.
In the software domain, there are strategies for handling these errors in such a way that they have no effect on the correct functioning of the system. These strategies are depicted in Figure D.3.

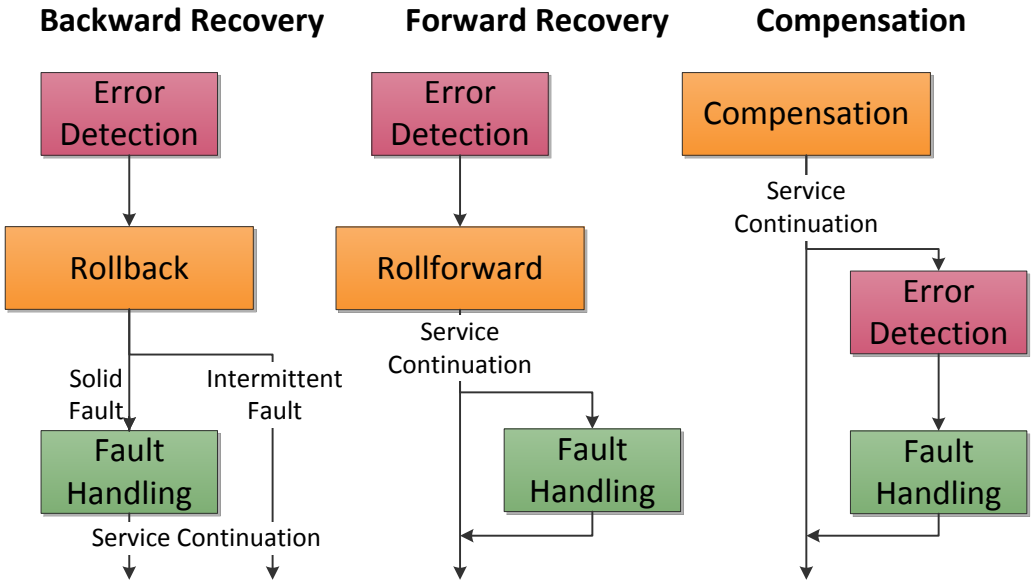

Figure D.3: *Fault tolerance strategies according to Avizienis et al. [4]*

These concepts are transferred to the AIMM system.

D.6.1 Fault Compensation

Model Name:	Model Kind:	Model Type:	ID:
Fault Compensation	VS-A5G	Guideline	G32
Addressed Aspects: VS-A5			
Description: Fault Compensation means that the *Skills* fulfill their *Task* even if errors occur. On the one hand, this goal is achieved by ensuring that the individual behaviors are robust against inaccuracies and small deviations. On the other hand, components are integrated into the process that neutralize possible errors. This enables the system to perform its *Task* correctly even if errors occur.			

Rationale 161: ***Handling uncertainties***
When implementing *Skills*, the expected errors should always be taken into account. The most common form of errors are small deviations from the exact values. This uncertainty must be absorbed by the *Skill* implementation. To do this, these small deviations must be prevented from accumulating. This can be achieved by referencing, for example.

Rationale 162: ***Independent of the error detection***
No error detection is required to apply the error compensation concept. Errors that occur have no influence on the result of the *Skill* as long as the errors are within the error tolerance of the compensation methods. However, the occurrence of errors reduces the robustness of the system. Without error detection, this happens unnoticed. In a system that is so overloaded, another small error can mean that the system can no longer perform the *Task* correctly. Error handling in this situation is complex, as the system failure is caused by a combination of different errors. The causes of these errors may have occurred much earlier and only have an effect in this situation through error propagation in combination with other errors. For larger systems, this can result in error handling not being feasible.

Rationale 163: ***Inefficiency and additional complexity***
The Fault Compensation deals with potential errors that often do not occur. This reduces efficiency and can even cause errors itself. The reason for this is the increased complexity of implementing the compensation strategy. The concept of Fault Compensation is therefore particularly suitable for frequently occurring errors, such as uncertainties. In addition, the complexity of the compensation strategies should be kept as low as possible so as not to generate more additional errors than are compensated for.

D.6.2 Backward Recovery

Model Name:	Model Kind:	Model Type:	ID:
Backward Recovery	VS-A5G	Guideline	G33
Addressed Aspects: VS-A5			
Description: The concept of Backward Recovery is to restore the system to the state it was in before the error occurred. Depending on the type of error, an attempt can then be made to restart the unmodified execution of the *Skills*. However, if the error is persistent, error handling must be carried out first.			

Rationale 164: ***Generic strategy***
The aim of the Backward Recovery concept is to return the system to the state it was in before the error occurred. The system then returns to its nominal state. The advantage of this strategy is that the recovery is universal. Using a checkpoint as a recovery state can serve as a fallback for many subsequent actions and the errors that may occur. The strategy is also generic: all actions performed since the error occurred must be undone. This applies to any action on any system.

Rationale 165: ***High efficiency during execution***
Since recovery is only carried out in the event of an error, execution does not lose efficiency during proper operation. However, effective error detection is absolutely essential.

Rationale 166: ***Oscillating behavior***
If errors occur deterministically, repeating the execution again leads to the identical error that triggered the Backward Recovery. This leads to the strategy being triggered again and thus to constantly repeating behavior. In the case of persistent errors, this oscillation can only be broken by successful error handling, i.e. neutralizing the error.

Rationale 167: ***Physical world cannot simply be reset***
The biggest challenge with the concept of Backward Recovery is to return the system to its previous state. This is not just about the state of the control flow, but about the state of the entire system and its environment. In contrast to pure software systems, a *Robot* and its environment cannot simply be reset due to the lack of controllability. For a Backward Recovery in robotics, the *Skills* of the system must be used to reset the state. Therefore, this recovery can be complex and often impossible in practice.

D.6.3 Forward Recovery

Model Name:	Model Kind:	Model Type:	ID:
Forward Recovery	VS-A5G	Guideline	G34
Addressed Aspects: VS-A5			
Description: With the *Forward Recovery* strategy, the problem is solved by applying an alternative strategy. In order to be able to apply this alternative strategy, error detection is necessary. As the faulty components are no longer used, troubleshooting is not mandatory. However, this reduces the redundancy and therefore the robustness of the system.			

Rationale 168: ***Strengths and weaknesses of different methods***
The Forward Recovery method is very suitable when different methods can solve the same problem but have different strengths and weaknesses. For example, one method can deliver results quickly, but these may be subject to errors. Another method is very reliable, but requires a high computational effort. Provided that the errors of the first method are reliably detected, the two methods can be combined using the *forward recovery* strategy to achieve high efficiency and reliability at the same time.

Rationale 169: ***Error handling can be avoided***
With the Forward Recovery strategy, troubleshooting is not mandatory even for persistent errors. The Forward Recovery method is therefore a good choice for errors that are difficult or impossible to resolve. For reasons of robustness, error correction should nevertheless be carried out where possible.

Rationale 170: ***Efficient in execution***
The Forward Recovery is the most efficient strategy in terms of execution. During the nominal process, Forward Recovery, like Backward Recovery, has only minimal impact on the process. In the event of an error, an alternative strategy can be used. It is not necessary to reset the state as with Backward Recovery, which has a positive effect on efficiency.

Rationale 171: ***Specific solution***
The main disadvantage of the Forward Recovery strategy is that it is very specific. The strategies are often manually tailored to a specific failure, system state and available *Skills*. The generalization of this recovery approach is an unsolved problem. Therefore, the use of Forward Recoveries greatly increases the complexity of the *Skill*. As these strategies are only used when errors occur, testing and maintaining the strategies is very time-consuming.

D.6.4 Fault Removal

Model Name:	Model Kind:	Model Type:	ID:
Fault Removal	VS-A5G	Guideline	G35
Addressed Aspects: VS-A5			
Description: The troubleshooting strategy consists of eliminating potential sources of error before they affect the system. This can be done either by the system itself or externally outside of operation. Fault Removal plays a particularly important role in the ongoing development process.			

Rationale 172: ***Reliable components***
In complex systems, the best solution is always to eliminate sources of error from the outset before the errors affect larger parts of the system. Reliable components are therefore the basis of every autonomous *Robot*.

Rationale 173: ***Complex causes of errors***
Even with perfect components, faults can be caused by interactions. It is therefore crucial for Fault Removal that the system provides an insight into these interactions.

D.6.5 Hierarchical Error Propagation

Model Name:	Model Kind:	Model Type:	ID:
Hierarchical Error Propagation	VS-M5A	Approach	A21
Addressed Aspects: VS-A5			
Description: If an error occurs in a state machine that cannot be handled locally, it is automatically escalated to the next hierarchy level. This continues until the error has been handled or the next higher level has been reached. This hierarchical error propagation ensures that errors are always handled at the correct level.			
Applies: G33, G34			

Rationale 174: ***As local as possible, as global as necessary***
Errors that occur should be handled as locally as possible in order to avoid error propagation as far as possible. However, if the error leaves the scope of a *Skill*, the problem must be solved at a different level. For example, a grasp *Skill* includes the ability to compensate for uncertainties and positioning errors in order to pick up an object. However, if the object has been removed, the grasp *Skill* cannot handle this error itself, but the error must be forwarded

to the higher-level *Skill*, e.g. the pick-up *Skill*. Further *Skills* can then be used at these higher levels, e.g. object localization, or the problem can be passed upwards again in order to continue with the next object, for example.
Each level has different ways of reacting to errors. If the error cannot be resolved at any level, execution is aborted.

Rationale 175: ***Backward recovery with hierarchical error propagation***
If a state of a higher hierarchy is restarted, this implies a reset of the process for the state machines contained in it, provided they are not the initial states. In the simplest case, a Backward Recovery can therefore be carried out by restarting an active state of the hierarchy. Of course, this only resets the state of the logic flow. Any changes in module states or the physical world are not reset. This must be monitored and, if necessary, implemented using additional routines, which increases complexity. This is usually still manageable for jumps that only comprise 1 or 2 levels. Choosing the hierarchy as the starting point for smaller Backward Recoveries is therefore a simple generic solution.

Rationale 176: ***Application of the forward recovery strategy***
With the Forward Recovery strategy, a problem must be solvable in two different ways. It makes sense to encapsulate each of these in a state machine. These two state machines are located together in one state. If an error occurs in the nominal solution, it is automatically propagated upwards and can then activate the alternative solution. However, this only works for arbitrary errors if the two solutions are completely independent of each other.

D.6.6 Execution Logging

Model Name:	Model Kind:	Model Type:	ID:
Execution Logging	VS-M5A	Approach	A22
Addressed Aspects: VS-A5			
Description: Execution logging plays a crucial role in the dependability of the *Robot*. Only by systematically recording all data during execution is it possible to trace error chains. This essentially includes sensor data, flow data and data within the sequence controller, module states such as the world model, communication between the modules and external interventions such as user interactions. In the AIMM system, all this information is stored as completely as possible during execution.			
Applies: G33, G34, G35			

Rationale 177: ***Error cause and error detection are often offset in time***
The most important point for intensive logging of all information in robotics is that error detection and error cause often occur at different times. When an error occurs, it often has

no immediate effect and is therefore difficult to detect. The existence of this error then only becomes apparent at another point in time.
For example, it is difficult to detect an error when localizing an object. If the *Robot* then grasps the object at a later point in time, various causes of error are possible. The stored information can then be used to check the various error causes, such as incorrect grasping strategy, calibration error, object has been removed, etc., and a suitable error handling strategy can be found.

Rationale 178: ***Handling large amounts of data and reproducible information***
The seamless recording of all data results in large amounts of data. The camera data and the information calculated from it, such as the depth data, are particularly important. As this data is easily reproducible, only the initial data could be stored. Other information could be reproduced as required. As today's storage media allow all data to be stored over many hours of operation, the reproducible data is also stored on the AIMM system.

Rationale 179: ***Middlewares, world model & process control***
The use of middleware, a central world model and central process control makes most of the information easily accessible. Most middlewares offer tools for logging communication. The two central modules, flow control via Rafcon and the world model, also offer mechanisms for logging. Logging the module statuses is more difficult. A separate solution must be found for each module. This is very time-consuming and therefore even with AIMM the complete status of each module is not recorded. However, this can often at least be reconstructed via the stored input and output data.

Rationale 180: ***Automatic labeled data***
The recorded data provides a lot of sensor data with contextual information. This can be used to optimize data-driven processes for the situations that arise. For example, an object position is verified after a successful pick-up process. This allows the object to be automatically marked in all camera images that were previously recorded so that the position of the *Robot* is known. In this way, large amounts of situation-relevant training data can be generated automatically.

D.6.7 RAFCON Error Handling and Logging

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Error Handling and Logging	VS-M5I	Implementation	I16

Addressed Aspects:
VS-A5

Description:
RAFCON offers various mechanisms for error handling. For example, exceptions from the Python level are catched by RAFCON states and mapped to the logical outcome aborted. If this is not linked, the state of the next hierarchy level is automatically left with outcome aborted. In this way, local error handling can be easily integrated or, alternatively, the problem is automatically passed on to the next hierarchy level.
A step-by-step mode is available for testing the *Skills*, in which each individual state is actively triggered. The data flows are also visualized. This provides an efficient tool for Fault Removal. State-based programming can thus be implemented and tested efficiently.
RAFCON automatically logs both the logical sequence and all data flows implemented in RAFCON for each execution. With the help of timestamps, error times can be precisely determined and, if necessary, synchronized with data from other sources. AIMM therefore uses the tools of AIMM to implement A21 and A22.

Implements:
A21, A22

Appendix E

AIMM Mission View

This chapter describes the *Mission View* of the AIMM system. It begins with an overview of all identified *Models* and their relationships with each other. The individual *Architecture Models* are then presented.

E.1 AIMM Mission Overview

***Guideline Models*:**

- Low Level Tasks
- High Level Tasks
- Continuous Abstraction Levels
- Worker Scaled Tasks
- 2-Phase Approach
- Setup Phase
- Execution Phase
- Production Scheduling
- Environmental Changes
- Static Mission Interface
- No Task Interfaces

***Approach Models*:**

- Task Primitives
- Skill-driven Task (De)composition
- Missions
- Industrial Robot Phases
- Trainee
- Production
- Static Mission
- Adaptive Tasks
- Control Mission Statemachines

***Implementation Models*:**

- RAFCON Mission Design
- RAFCON Mission Phases
- RAFCON Execution
- RAFCON GUI

E.2 AIMM Mission Abstraction

E.2.1 Low Level Tasks

Model Name:	Model Kind:	Model Type:	ID:
Low Level Tasks	VM-M1G	Guideline	G36
Addressed Aspects: VM-A1			
Description: A fundamental concept of AIMM is to keep the abstraction level of the *Tasks* as low as possible. The aim is to identify and explicitly name these smallest possible *Tasks*. As a rule, these are created by decomposing *Tasks* of higher levels of abstraction.			

Rationale 181: *Smallest possible Tasks*
According to Definition 3.3, *Tasks* represent a targeted modification to the physical environment or the extraction of information from it. The smallest possible *Task* must of course also satisfy this definition.
No algorithm can change the physical world, it is always an interplay of hardware and

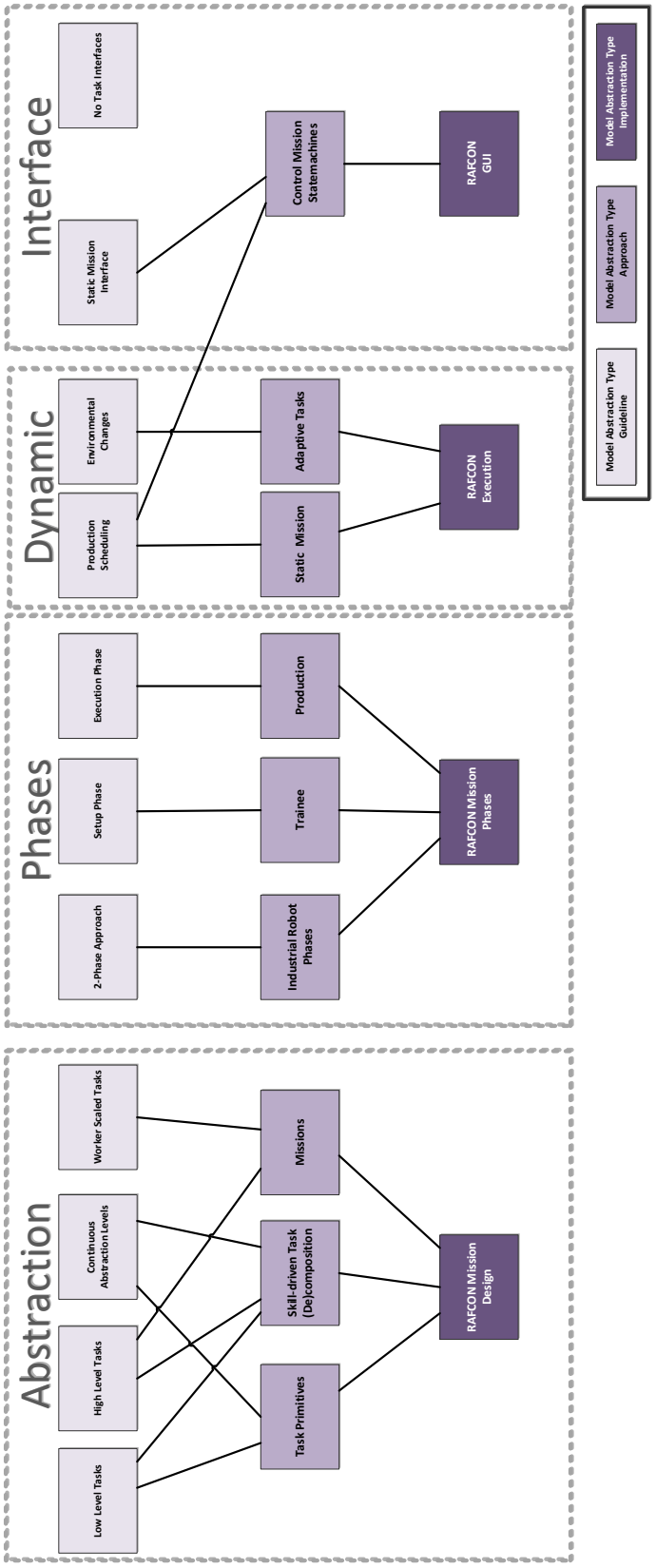

Figure E.1: *The Relations of of the Mission View Models*

software. Ultimately, the *Robot* can only change the physical world through its actuators. The smallest possible *Tasks* are therefore individual movements of the *Robot*. Theoretically, every movement in the continuous world can be broken down into an infinite number of partial movements.

For perception *Tasks*, i.e. the extraction of information from the physical world, it is more difficult to identify the finest granularity, as this cannot be reduced to individual sensor measurements. Since this is a passive process, no *Task* can generally be defined. For example, a new image may not contain any additional information or may contain a lot of different information. Other information can only be extracted from several images, e.g. the speed of an object. The smallest possible perception *Task* is therefore abstractly the smallest information about the physical environment that can be obtained. In contrast to the active *Tasks*, this cannot be broken down into infinitely small parts, as the information is limited, e.g. in the sensor data.

Rationale 182: ***Explainability of the behavior***
Tasks are generally independent of the *Robot*'s actions. A specification as a *Task* makes it possible to observe the behavior of the *Robot* in the physical world or in the world model. The finer the granularity of the *Tasks*, the more precisely the *Robot*'s behavior can be observed, as it is defined what the current goals are.
If the *Robot* cannot achieve a goal, this also directly explains to the observer why the *Robot* adapts its behavior. At a lower level of abstraction, this also alters the desired goals, which in turn can be observed.

E.2.2 High Level Tasks

Model Name:	Model Kind:	Model Type:	ID:
High Level Tasks	VM-M1G	Guideline	G37
Addressed Aspects: VM-A1			
Description: High level *Tasks* describe *Tasks* at a very high level of abstraction. This means that the state of the physical world is specified without specifying how this is to be achieved. An example of a high-level *Task* is: "Tidy up the room". Due to the high level of abstraction, high-level *Tasks* are usually very independent of the specific *Robot*. In addition, high-level *Tasks* are often also very independent of the current state of the physical world.			

Rationale 183: ***Operational freedom***
The specification of *Tasks* at a high level gives the *Robot* a freedom of how solve the *Task*. The *Robot* can recognize the necessary sub-steps itself and adapt them if necessary.

Rationale 184: ***Avoidance of overspecification***
A high level of abstraction avoids over-specification of *Tasks*. Since the *Task* describes the target state without specifying the way to get there, no *Tasks* are specified that are not necessary for solving the problem.

Rationale 185: ***More independent of the environment and its state***
Due to the high level of abstraction, high-level *Tasks* are also less dependent on the specific environment and the current state of the environment. A "Clean up the room" *Task* makes sense regardless of the current state of the room. Of course, the implementation looks very different. The *Task* is also very independent of the room itself, still "Clean up the room" makes sense in any room.

Rationale 186: ***Independent of the Robot and its state***
A high degree of abstraction also decouples the *Task* from the *Robot* system and its current state. The *Task* of "tidying up the room" can be delegated to any *Robot* that is suitable for the *Task* in principle. Tasks at this level can often also be transferred to humans, which can be useful for collaborative applications.

E.2.3 Worker Scaled Tasks

Model Name:	Model Kind:	Model Type:	ID:
Worker Scaled Tasks	VM-M1G	Guideline	G38
Addressed Aspects: VM-A1			
Description: The goal of AIMM is to take over *Tasks* that are currently performed by human workers. The system must therefore be able to process *Tasks* at the level of abstraction used by human workers.			

Rationale 187: ***Different levels of abstraction***
It can be seen that workers in factories are assigned *Tasks* with different levels of abstraction. This depends heavily on the complexity of the *Task* and the possible dangers or effects of errors. If the *Task* is simple and non-critical, the level of abstraction is often very high. For complex or dangerous *Tasks*, on the other hand, very specific protocols often have to be processed.

Rationale 188: ***Simple Tasks***
Simple *Tasks* are defined in industry at a high level of abstraction. A typical example is tidying up the workplace. It is not specified what needs to be cleared away in which order, what is waste, etc. AIMM must therefore be able to handle abstraction at this level.

Rationale 189: ***Complex or dangerous Tasks***
For complex *Tasks* or *Tasks* with far-reaching consequences in the event of errors, the processes in the industry are precisely specified, e.g. assembly sequence, tightening torques, tools, etc. The AIMM system must therefore also be able to perform *Tasks* at this level.

E.2.4 Continuous Abstraction Levels

Model Name:	Model Kind:	Model Type:	ID:
Continuous Abstraction Levels	VM-M1G	Guideline	G39
Addressed Aspects: VM-A1			
Description: The concept of Continuous Abstraction Levels means that *Tasks* are not assigned to a specific abstraction level. Instead, the *Task* abstraction levels of the *Tasks* form a continuous spectrum from low-level *Tasks* to very abstract high-level *Tasks* that define the *Mission*.			

Rationale 190: ***Continues vs. layered abstraction levels***
There is the approach of organizing *Task* abstraction levels in layers. For example, some *Tasks* form a layer in which production steps are recorded, e.g. “Assemble housing”, an underlying layer defines the *Tasks* that must be executed for this, e.g. “Screw screw A into thread B” and then possibly another layer that represents the individual *Robot* actions, “Move to approach frame A”. This approach makes it possible to structure complex *Robot* programs. However, it also severely restricts the modeling options. For example, two pre-assembled parts can be joined together. A fixed hierarchy of 3 does not allow these *Tasks* to be specified as superordinate. Another problem is that sequenced *Tasks* should always be at the same level of abstraction. A mixture of high-level *Tasks* and *Tasks* at a lower level of abstraction is therefore difficult to represent. In the industrial sector in particular, however, there are such *Tasks* for workers with a strong mixture of abstraction levels. AIMM therefore does not use a fixed classification of abstraction levels. Any *Tasks* can therefore be explicitly represented and linked to each other. This leads to a continuous spectrum of *Task* abstraction levels.

E.2.5 Task Primitives

Model Name:	Model Kind:	Model Type:	ID:
Task Primitives	VM-M1A	Approach	A23
Addressed Aspects: VM-A1			
Description: Task primitives represent the lowest level of abstraction in the AIMM system. This indicates that a *Task* primitive does not contain any *Subtasks*. By applying the G39 concept, no fixed abstraction level is defined for *Task* primitives. *Task* primitives can therefore exist at different abstraction levels. It is therefore also true that all *Tasks* can be broken down into *Task* primitives.			
Applies: G36, G39			

Rationale 191: *Tasks primitives are not smallest possible Tasks*
As discussed in G36, *Tasks* can theoretically be broken down into very small *Subtasks*. However, such a decomposition has no practical benefit, but creates additional complexity. The *Task* primitives with the lowest level of abstraction are therefore individual movement commands for active *Tasks*. For the perception *Tasks*, the lowest level of abstraction is a single piece of information that is added to the world model.

Rationale 192: *Contains no Subtasks*
Contains no *Subtasks* can be achieved by completely decomposing the *Task* into the smallest possible *Task* primitives. However, many *Task* primitives are at a much higher level of abstraction. There can be various reasons for this. For example, if a *Task* can be solved by a control loop, it is usually not broken down any further.

E.2.6 Skill-driven Task (De)composition

Model Name:	Model Kind:	Model Type:	ID:
Skill-driven Task (De)composition	VM-M1A	Approach	A24
Addressed Aspects: CV-C1			
Description: The *Skills* of the AIMM system are used to break down the *Tasks*. Each *Skill* can solve *Tasks*. If the *Skill* contains *Subskills*, these also solve *Tasks*. The use of *Skills* therefore leads to *Task* decomposition. However, this often only happens at runtime, as the specific *Subtasks* are only determined based on the current situation. The structure of the *Skill* defines the abstraction levels of the *Tasks*.			
Applies: G36, G37, G39			

Rationale 193: ***General relation between Tasks an Skills***
First, a few general observations about the relationship between *Tasks* and *Skills*

- **One Skill can solve multiple Tasks** A *Skill* can usually solve multiple *Tasks*.
- **One Task can be solved by multiple Skills** A certain *Task* can possibly be solved by multiple *Skills*.
- **Every Task primitive has to be solved by a Skill** Every *Task* primitive has to be solved by at least one *Skill*. This *Skill* must be a *Skill Primitive* as only these do not contain *Subskills*.

Rationale 194: ***The Task decomposition depends on the situation***
Even if the same *Skill* is used for a *Task*, the *Task* decomposition is usually not identical. This is because the *Tasks* of the *Skills* are generated depending on the situation. The actual *Task* decomposition therefore only takes place at runtime.

Rationale 195: ***Continues Task Abstraction Levels***
The use of *Skills* to decompose *Tasks* automatically leads to a complex structure of *Tasks* at very different levels of abstraction. Autonomous systems with very complex *Skills* result in an almost continuous spectrum of *Task* abstractions.

E.2.7 Missions

Model Name:	Model Kind:	Model Type:	ID:
Missions	VM-M1A	Approach	A25
Addressed Aspects: CV-C1			
Description: The missions form the highest level of abstraction. This is where the top-level *Tasks* and their dependencies are specified. Missions can be structured hierarchically and thus specify a fixed *Task* decomposition.			
Applies: G38, G37			

Rationale 196: ***Top-Level Abstraction - Independent from system and situation***
In AIMM, the missions and the *Tasks* they contain form the highest level of abstraction of the system. Missions are therefore generally independent of the *Robot* and the current situation. For example, the same mission could also be assigned to a human worker.
The current state of the environment also usually only plays a very subordinate role at this level of abstraction.

Rationale 197: ***Executability of missions***
Missions can only be executed by the *Robot* if the system has the *Skills* to execute them. It should be noted that only the lowest level of abstraction must be executable for hierarchical missions.

E.2.8 RAFCON Mission Design

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Mission Design	VM-M1I	Implementation	I17
Addressed Aspects: VM-A1			
Description: Missions are implemented for AIMM with RAFCON. *Tasks* are represented by states and dependencies by logical links. As *Tasks* are specific, no data flows are required at this level. Hierarchical *Tasks* are represented by hierarchy states.			
Implements: A23, A24, A25			

Rationale 198: ***Same tool for Skills an Tasks***
At AIMM, the same software tool RAFCON is used to implement both *Skills* and *Missions*. On the one hand, this is practical because it is very easy to link *Tasks* with *Skills*. Appropriately parameterized *Skills* are used for the lowest abstraction level of the mission. However, there is a risk of *Tasks* and *Skills* being confused with each other, as they initially look very similar in the implementation. However, a state that represents a *Task* never receives information about a data port. A state can only be a *Task* if:

1. only *Subtasks* are contained
2. no *Capabilities* as substates
3. no internal data flows

E.3 AIMM Mission Phases

E.3.1 2-Phase Approach

Model Name:	Model Kind:	Model Type:	ID:
2-Phase Approach	VM-M2G	Guideline	G40
Addressed Aspects: VM-A2			
Description: Autonomous systems solve their *Tasks* themselves by definition. To do this, however, the *Robot* must have information or be in a certain state. The concept of the 2-phase approach therefore divides the *Mission* into two phases. One phase, the setup phase, which serves to collect information and establish the required system state, and a phase in which the actual *Tasks* are solved. The phase structure of the two-phase approach therefore consists of first collecting information and checking the system status in a setup phase. This is followed by the transition to the execution phase. This is only exited again when the conditions or the system status have changed.			

Rationale 199: ***Autonomy in both phases***
The separation into two phases does not mean that the *Robot* is not autonomous in the first phase. Here too, the system solves *Tasks* independently. However, these *Tasks* are not specified externally, but by the system

Rationale 200: ***What are the prerequisites for the 2-phase approach***
The division into 2 phases is possible if the information and system states are then valid for a

longer period of time. If these change more frequently, these processes must be integrated into the actual execution.

E.3.2 Setup Phase

Model Name:	Model Kind:	Model Type:	ID:
Setup Phase	VM-M2G	Guideline	G41
Addressed Aspects: VM-A2			
Description: The purpose of the setup phase is to enable the *Robot* to solve *Tasks*. The *Tasks* that are solved in the setup phase are therefore not specified externally, but come from the system itself. In general, the setup phase can be divided into the categories system state, physical environment and mission environment.			

Rationale 201: *System state*
System states are *Tasks* that collect information about the *Robot* itself or change the state of the system. These can be calibration routines or a tool change, for example.

Rationale 202: *Physical Environment*
The physical environment can be explored in the setup phase. A typical example would be the creation of a map of the environment.

Rationale 203: *Mission Environment*
The mission environment can also be explored during the setup phase. E.g. which button must be pressed to start a certain machine.

E.3.3 Execution Phase

Model Name:	Model Kind:	Model Type:	ID:
Execution Phase	VM-M2G	Guideline	G42
Addressed Aspects: VM-A2			
Description: In the execution phase, the *Tasks* assigned to the system by the *Mission Environment* are solved. This is based on the information and configurations from the setup phase. The productive use of the system therefore takes place exclusively in the Execution phase. As long as the *Mission* and physical environment remain constant and the system status meets the requirements, the system remains in the execution phase.			

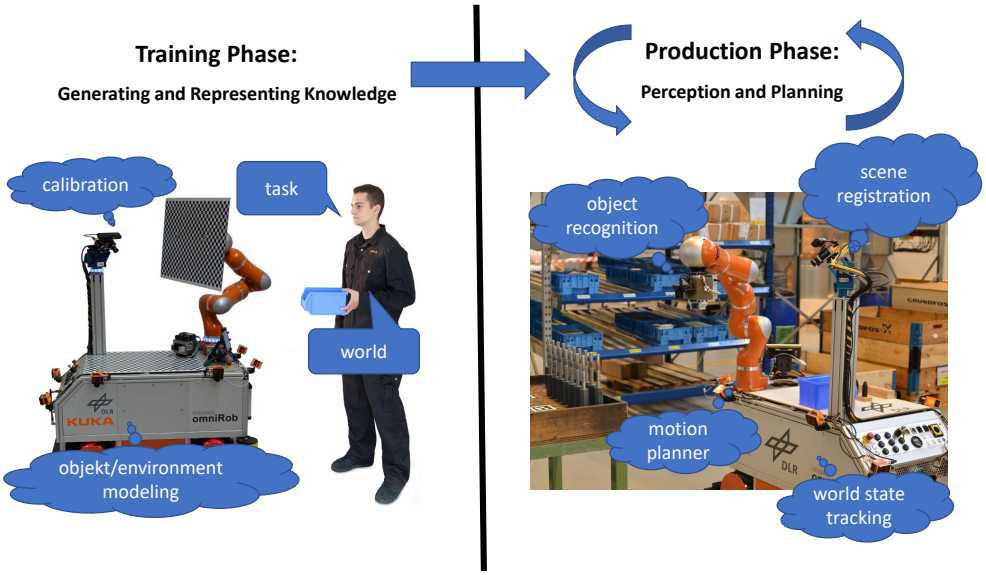

Figure E.2: *The two phases of a industrial Robot's Mission. (source: Dömel et al. [35])*

Rationale 204: ***Monitoring of setup***
Another internal *Task* in the execution phase can be the monitoring of information and configuration from the setup phase.

E.3.4 Industrial Robot Phases

Model Name:	Model Kind:	Model Type:	ID:
Industrial Robot Phases	VM-M2A	Approach	A26
Addressed Aspects: VM-A2			
Description: A *Mission* in an industrial context usually has a 2-phase structure. This is depicted in Figure E.2. The setup phase is called training and the execution phase is called production.			
Applies: G40			

E.3.5 Trainee

Model Name:	Model Kind:	Model Type:	ID:
Trainee	VM-M2A	Approach	A27

Addressed Aspects:
VM-A2

Description:
In the training phase, the *Robot* has to learn the sequence of *Tasks*, which is very similar to training a human worker. In addition to the information about what to do, the *Robot* system also needs information about the world. The term world is used as an internal representation of the *Robot*'s *Physical Environment*. The world can be very abstract, e.g. defined workstations, but geometric models of the environment for path planning, for example, can also be part of the concept. This knowledge must be acquired during the setup phase, possibly with the support of a human employee. Robot-specific *Tasks* that are independent from the actual mission, such as camera calibration, are also carried out during the setup phase.

Applies:
G41

E.3.6 Production

Model Name:	Model Kind:	Model Type:	ID:
Production	VM-M2A	Approach	A28

Addressed Aspects:
VM-A2

Description:
In the production phase, the *Robot* carries out the *Tasks* of the *Mission* autonomously. For the external observer, e.g. the owner of the factory, this is the productive phase. In production facilities, there are two categories of common *Tasks* that are difficult to automate with traditional automation approaches and are usually performed by human workers. These are pick-up and delivery *Tasks* and assembly *Tasks*, which are therefore the focus of AIMM.
Pick-up and delivery *Tasks* include all *Tasks* in which a part has to be picked up from one location and brought to another location. Classic *Tasks* in industry are warehouse logistics, e.g. restocking parts magazines at the workplace from the warehouse.
Assembly *Tasks* are all *Tasks* in which parts have to be assembled. These are usually very simple operations such as loading machines or pre-assembly for automated or human workstations, e.g. inserting screws into drilled holes. As a rule, these are not complex *Tasks*, such as the assembly of a complete pump. Even with human labor, such *Tasks* are prone to errors and are therefore split into individual steps.

Applies:
G42

E.3.7 RAFCON Mission Phases

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Mission Phases	VM-M2I	Implementation	I18
Addressed Aspects: VM-A2			
Description: RAFCON is used to implement the *Mission* phases in AIMM. At least one state machine is implemented for each phase. They do not differ technically.			
Implements: A27, A28			

E.4 AIMM Mission Dynamic

E.4.1 Production Scheduling

Model Name:	Model Kind:	Model Type:	ID:
Production Scheduling	VM-M3G	Guideline	G43
Addressed Aspects: VM-A3			
Description: The dynamics of the *Mission* result from the dynamics that occur in production during resource planning. Even if this planning, supported by software, is becoming ever faster and more efficient, the resulting plans remain constant over longer periods of time. Ultimately, even human workers cannot react to changes in plans as quickly as they like. The Production Scheduling concept therefore assumes a dynamic that expects changes and adjustments in hours or days.			

E.4.2 Environmental Changes

Model Name:	Model Kind:	Model Type:	ID:
Environmental Changes	VM-M3G	Guideline	G44
Addressed Aspects: VM-A3			
Description: In an industrial environment, the basic structure is constant. As a rule, the things relevant to the *Task*, such as workstations, machines and warehouses, remain in the same place. At the same time, the details of the environment are constantly changing. Materials and preliminary products are transported and temporarily stored. The environment is constantly changing, especially at the workstations for human workers. The elements can therefore be regarded as static at a high level of abstraction. At the lower levels of abstraction, however, quasi-static changes are to be expected.			

E.4.3 Static Mission

Model Name:	Model Kind:	Model Type:	ID:
Static Mission	VM-M3A	Approach	A29
Addressed Aspects: VM-A3			
Description: In line with the concept of G38, a high degree of abstraction is to be expected for AIMM. At this level of abstraction, a static mission can be assumed based on G43 and G44. This means that no changes to the mission itself are assumed during the mission.			
Applies: G43			

E.4.4 Adaptive Tasks

Model Name:	Model Kind:	Model Type:	ID:
Adaptive Tasks	VM-M3A	Approach	A30
Addressed Aspects: VM-A3			
Description: Modifications are to be expected at the lower levels of abstraction as the physical environment changes. For AIMM, adaptive *Tasks* are therefore used at the lower levels of abstraction. These make it possible to react flexibly to changes in the environment. This also means that the *Tasks* on the lower abstraction levels are only defined at runtime.			
Applies: G44			

E.4.5 RAFCON Execution

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Execution	VM-M3I	Implementation	I19
Addressed Aspects: VM-A3			
Description: The *Mission* is executed using the RAFCON framework. This makes it possible to represent the static missions in the high-level area as *Task* state machines. The *Skill* concept, which is also implemented in RAFCON, is used to implement the adaptive *Tasks*. This decides which specific *Tasks* are executed based on the current situation.			
Implements: A29, A30			

E.5 AIMM Mission Interface

E.5.1 Static Mission Interface

Model Name:	Model Kind:	Model Type:	ID:
Static Mission Interface	VM-M4G	Guideline	G45
Addressed Aspects: VM-A4			
Description: A *Mission Control Interface* for a static mission is limited to starting, stopping and monitoring the *Mission* progress. It is not possible to change the sequence itself via the interface. To realize the monitoring, the interface must communicate the status of the execution.			

E.5.2 No Task Interfaces

Model Name:	Model Kind:	Model Type:	ID:
No Task Interfaces	VM-M4G	Guideline	G46
Addressed Aspects: VM-A4			
Description: As the missions do not provide for cross-system *Tasks*, there are no *Task Interfaces*.			

E.5.3 Control Mission Statemachines

Model Name:	Model Kind:	Model Type:	ID:
Control Mission Statemachines	VM-M4A	Approach	A31

Addressed Aspects:
VM-A4

Description:
The *Mission* is implemented as a state machine. This does not change during execution, as the *Mission* itself is static. Control of the state machine is therefore limited to starting, pausing or stopping. When starting, the state machine is activated in its initial state. When pausing, all transitions between the states are frozen. The state can therefore no longer be changed. However, actions that take place within a state are not paused. When stopping, the current state is exited. It is therefore not possible to resume execution directly at this point. The currently active states are displayed to monitor the process.

Applies:
G45

E.5.4 RAFCON GUI

Model Name:	Model Kind:	Model Type:	ID:
RAFCON GUI	VM-M4I	Implementation	I20

Addressed Aspects:
VM-A4

Description:
The RAFCON software is used to control and monitor these *Missions*. This enables the execution of state machines. A special feature of RAFCON is that the initial state can be freely selected. RAFCON also offers the option of pausing, restarting or aborting the execution. The current state of the state machine is displayed both in the graphical representation of the state machine and in the text output.

Implements:
A31

Appendix F

ARDEA Physical View

This chapter describes the *Physical View* of the ARDEA system. It begins with an overview of all identified *Models* and their relationships with each other. The individual *Architecture Models* are then presented.

F.1 ARDEA Physical Overview

***Guideline Models*:**

- Fail-safe Redundancy
- Indoor Usecases Size
- FPGA Processing
- Local Computing
- Dedicated Realtime Computer
- Make Everything Perceivable
- Exchangable Components
- Limited Power Resources
- Unlimited Power Resources
- Safety Operator
- Low Level Motion Commands

***Approach Models*:**

- FPGA Stereo Processing
- IT Setup
- Wide Angle Stereo Cameras
- Sensor Composition
- Frame & Stack
- Propulsion Frame
- Power Setup
- Safety Pilot
- Remote Controller

***Implementation Models*:**

- ARDEA Stack IT
- ARDEA Stack Sensor
- ARDEA Frame
- ARDEA Stack Power
- Ardea Remote Controller

F.2 ARDEA Physical Structure

F.2.1 Fail-safe Redundancy

Model Name:	Model Kind:	Model Type:	ID:
Fail-safe Redundancy	VP-M1G	Guideline	G1
Addressed Aspects: VP-A1			
Description: A central design concept of ARDEA is that the system remains airworthy even if a rotor fails. The structure of the drone must therefore provide appropriate redundancy.			

Rationale 1: ***Loss of efficiency due to redundancy***
Redundancy is always accompanied by a loss of efficiency, i.e. the system could carry more payload, fly longer or be more compact without redundancy.

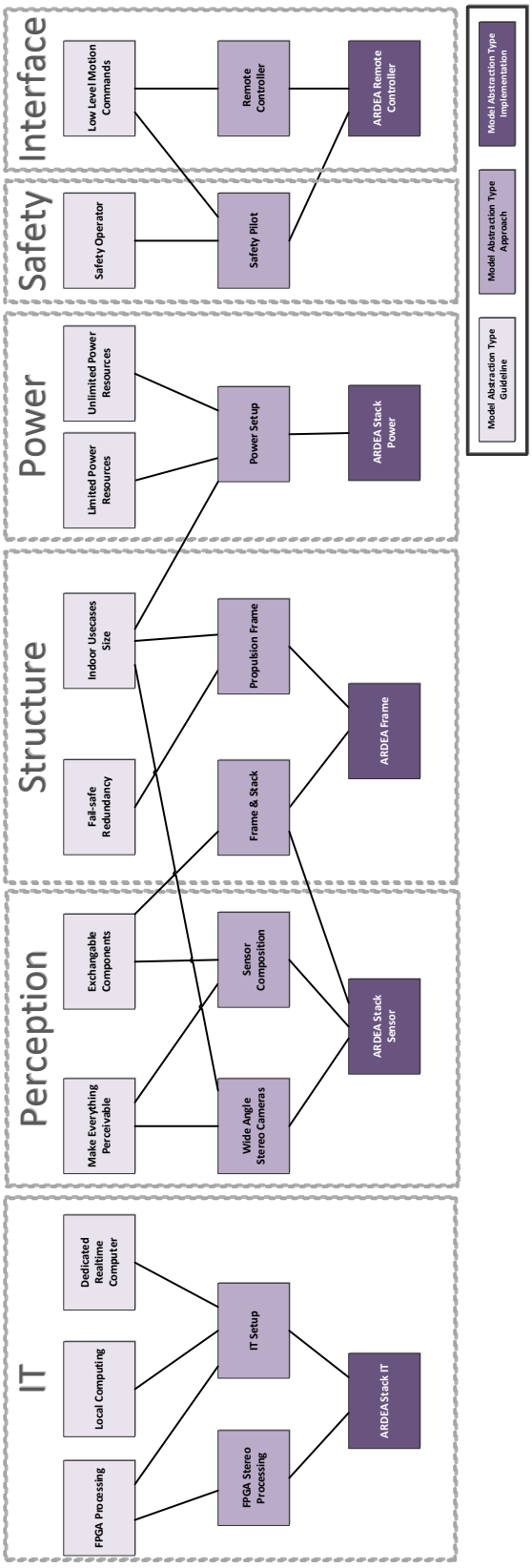

Figure F.1: *The Relations of the ARDEA Physical View Models*

Rationale 2: ***Missions outside the laboratory***
The ARDEA system is also intended to be used outside the laboratory environment in analog missions. The safety measure for these missions is the safety pilot. For this concept to work, a reliable flight capability of the system is crucial. ARDEA is therefore equipped with redundant rotors.

F.2.2 Indoor Usecases Size

Model Name:	Model Kind:	Model Type:	ID:
Indoor Usecases Size	VP-M1G	Guideline	G2
Addressed Aspects: VP-A1			
Description: In addition to use in open areas, ARDEA can also be used in caves or houses. This limits the maximum size of the system as otherwise doors or windows can no longer be passed.			

Rationale 3: ***Limitation of size leads to limitation of payload***
Limiting the size of the system practically limits the payload of a multicopter. This is a major challenge for an autonomous aircraft, as sensors and IT components are not infinitely scalable. The system should therefore be designed to be as large as possible.

F.2.3 Frame and Stack

Model Name:	Model Kind:	Model Type:	ID:
Frame and Stack	VP-M1A	Approach	A1
Addressed Aspects: VP-A1			
Description: ARDEA's approach is to separate the system into two components, the frame and the stack. The frame implements the propulsion of the system, the stack contains the sensors, the computers and the power supply. Replacing the frame or stack is designed in such a way that it can be carried out without a significant amount of effort. This allows damage to be repaired quickly and the system to be reconfigured according to requirements.			
Applies: G4			

Rationale 4: ***Granularity of Components***
With two components, the granularity is very high. Of course, the individual elements can be exchanged. However, larger functional units are not easily interchangeable.

Rationale 5: ***Number of interfaces***
The electrical interface between the stack and frame is limited to the power supply and the bus line to the controller. If, for example, the computers were to be disconnected from the sensors, many connections would have to be plugged in, which can lead potentially to errors and make replacement time-consuming.

Rationale 6: ***Less compact and optimized***
A component design leads to a less compact design. This is because the individual components must remain accessible. In addition, the individual elements can no longer be freely positioned. This has a negative effect on the cable lengths, but also on the positioning of the sensors

F.2.4 Propulsion Frame

Model Name:	Model Kind:	Model Type:	ID:
Propulsion Frame	VP-M1A	Approach	A2
Addressed Aspects: VP-A1			
Description: The frame is the basis for ARDEA's propulsion system. It also determines the shape of the *Robot*. An equilateral triangle was chosen to keep the system compact. To achieve redundancy, there are two rotors at each corner of the triangle. This creates space inside the frame and enables the positioning of sensors.			
Applies: G1, G2			

Rationale 7: ***Protective function of the frame***
The frame forms a ring around the electronic components. In the event of contact with the environment, it is therefore usually the frame that absorbs the most energy. The sensitive sensors and IT components are protected inside the system.

Rationale 8: ***Simple, lightweight and robust construction***
In principle, the frame consists of just three bars and connectors and motor supports in the corners. Cross bracing or similar for reinforcement is not required. This makes the frame simple, but also light and robust.

F.2.5 ARDEA Frame

Model Name:	Model Kind:	Model Type:	ID:
ARDEA Frame	VP-M1I	Implementation	I1
Addressed Aspects: VP-A1			
Description: The ARDEA frame consists of three carbon tubes that form an equilateral triangle with aluminum connectors. The rotors are mounted axially in pairs on the connectors. The motor controller and the *Robot*'s landing gear are also located on the frame.			
Implements: A1, A2			

Rationale 9: ***Axial rotor mounting***
Axial rotor mounting has advantages, but also disadvantages. Axial mounting allows the rotors to be mounted compactly, which keeps the flight system smaller. Another advantage is that the cameras have a better, unobstructed viewing angle. As the rotors rotate in opposite directions in pairs, the system is also inherently rotationally stable. The disadvantage of axial mounting is that the lower rotor operates in the downstream of the upper rotor. This reduces the efficiency and the maximum thrust of the lower rotor.

F.3 ARDEA Physical Perception

F.3.1 Make Everything Perceivable

Model Name:	Model Kind:	Model Type:	ID:
Make Everything Perceivable (ARDEA)	VP-M2G	Guideline	G3
Addressed Aspects: VP-A2			
Description: The system's ability to perceive is an important part of the interface between the *Robot* and the physical environment. Perception is necessary to recognize uncertainties or erroneous information and to monitor processes. The more information the system has at its disposal, the better the *Robot* can react. The concept is therefore to make as much information as technically possible perceptible. It follows that the system should be equipped with many sensors.			

Identical to AIMM G3

F.3.2 Exchangable Components

Model Name:	Model Kind:	Model Type:	ID:
Exchangable Components (ARDEA)	VP-M2G	Guideline	G4
Addressed Aspects: VP-A1, VP-A2			
Description: The mechanical exchangeability of components is an important concept for ARDEA. The aim for ARDEA is to be able to replace both the propulsion system and the sensors without any major effort.			

Similar to AIMM G4

Rationale 10: ***Replacement of the propulsion system***
Replacing the propulsion system is primarily a way of repairing the system quickly. No individual parts need to be replaced, but a functioning and tested component can be replaced without much effort. However, the components are technically identical.

Rationale 11: ***Replacement of the sensors***
The interchangeability of the sensors is used to reconfigure the system. This means that specific sensors are only required for certain missions. As the weight of a flight system is severely limited, the system can be optimally equipped for the *Task* at hand without carrying unnecessary weight.

F.3.3 Wide Angle Stereo Cameras

Model Name:	Model Kind:	Model Type:	ID:
Wide Angle Stereo Cameras	VP-M2A	Approach	A3
Addressed Aspects: VP-A1, VP-A2			
Description: ARDEA's approach of making everything perceptible is achieved through the use of cameras. Mechanics such as a pan-tilt unit are not used for reasons of weight, among others. Therefore, several cameras with wide-angle lenses are used. These can cover a very wide angle of view with a low weight.			
Applies: G2, G3			

Rationale 12: ***Sensor adjustment through Robot movement***
The sensors are fixed to the system. However, the viewing angle around the vertical axis of the system can be modified by turning the *Robot*. This is not possible for the downward or upward coverage, at least in the static case. The aperture angles of the cameras must therefore be selected so that they cover at least 90 degrees upwards and downwards.

F.3.4 Sensor Composition

Model Name:	Model Kind:	Model Type:	ID:
Sensor Composition	VP-M2A	Approach	A4
Addressed Aspects: VP-A1, VP-A2			
Description: The core sensors of the ARDEA system consist of the wide-angle cameras and the IMU. These sensors are used in combination to estimate the position of the system, so the sensors must be calibrated to each other. On the hardware side, it is therefore important that the cameras are fixed to each other and to the IMU. For ARDEA, all sensors are therefore mounted on a stack. Additional sensors can also be mounted on this stack.			
Applies: G3, G4			

Rationale 13: ***Calibrated stack***
As all sensors are mounted on the stack, the calibration of the system is independent of the frame. No new calibration is therefore required if the stack is replaced.

F.3.5 ARDEA Stack Sensor

Model Name:	Model Kind:	Model Type:	ID:
ARDEA Stack Sensor	VP-M2I	Implementation	I2

Addressed Aspects:
VP-A2, VP-A3

Description:
Two pairs of stereo cameras with wide-angle lenses are mounted on the ARDEA stack. These are aligned in the same way in the horizontal plane and allow a free view between two pairs of rotors. In the vertical plane, the stereo camera pairs are offset by 60 degrees. One pair has the optical axis directed 60 degrees downwards, the other 60 degrees upwards. Together, the cameras cover a range of 240° in the vertical plane. The IMU is mounted in the center of the stack with vibration damping.

Implements:
A3, A4

Rationale 14: ***Perception of the surroundings***
The system's sensor technology makes it possible to capture the entire environment in RGB as well as the depth data by rotating around the vertical axis. This allows the flight system to very quickly create a geometric representation of the surroundings.

Rationale 15: ***Monitoring the flight altitude***
Thanks to the alignment of the stereo cameras, the distance to the ground and the distance to the ceiling are always recorded. This reliably prevents collisions.

Rationale 16: ***Maximum disparity limits 3d perception***
The quality of the stereo depth data decreases with the distance to the *Robot*. The base distance and the resolution of the cameras allow reliable depth information up to a distance of approximately 4m. The implementation on Ardea is not designed for larger distances or higher flight altitudes.

F.4 ARDEA Physical IT

F.4.1 Local Computing

Model Name:	Model Kind:	Model Type:	ID:
Local Computing (ARDEA)	VP-M3G	Guideline	G5
Addressed Aspects: VP-A3			
Description: A mobile system will change its location. In order to have a reliable data connection at all times, a large amount of technical effort is required. The concept of local computing therefore ensures that all computations are carried out on the system itself. This significantly reduces the infrastructure requirements.			

Identical to AIMM G6

F.4.2 Dedicated Realtime Computer

Model Name:	Model Kind:	Model Type:	ID:
Dedicated Realtime Computer	VP-M4G	Guideline	G6
Addressed Aspects: VP-A3			
Description: The system uses a dedicated computer exclusively for time-critical processes. The implementation of this concept at hardware level significantly simplifies the reliable operation of the system.			

Rationale 17: ***Native control of the actuators***
With a dedicated computer for real-time-critical components, care can be taken to ensure that the corresponding interfaces are already available natively. This reduces the complexity of the system.

Rationale 18: ***Specific requirement***
Software components with real-time requirements are very different from other components. For example, communication must be guaranteed securely and at a high frequency. However, the computing power requirements of these components are often low. Therefore, smaller processors can often also meet these requirements.

F.4.3 FPGA Processing

Model Name:	Model Kind:	Model Type:	ID:
FPGA Processing	VP-M4G	Guideline	G7
Addressed Aspects: VP-A3			
Description: For suitable applications, FPGAs offer the possibility of solving computationally intensive tasks very quickly and efficiently at the same time. Compared to alternative IT components such as GPUs or processors, this saves installation space, weight and energy. At ARDEA, an FPGA is therefore integrated into the IT hardware.			

Rationale 19: ***FPGA as ASIC***
In ARDEA, the FPGA is operated by the function as an ASIC. This means that the flexibility of the FPGA is used to implement a specific algorithm. A flexible adaptation of the algorithms or the addition of further algorithms is not carried out due to the high implementation effort.

Rationale 20: ***High frequency and parallelization***
The use of FPGAs is worthwhile for problems that occur frequently, place high demands on computing time and can be parallelized. In robotics, this applies to many components of the perception pipeline.

F.4.4 IT Setup

Model Name:	Model Kind:	Model Type:	ID:
IT Setup	VP-M3A	Approach	A5
Addressed Aspects: VP-A3			
Description: The computer structure of the ARDEA system consists of three components. An FPGA for processing the stereo data. A high-level computer with good computing power for carrying out computationally intensive operations. The cameras are connected to the high-level computer. A low-level computer for tasks with real-time requirements. The IMU and the motor controller are connected to the low-level computer. The high level computer is connected to the other two components via an Ethernet interface.			
Applies: G6, G5, G7			

F.4.5 Stereo Processing FPGA

Model Name:	Model Kind:	Model Type:	ID:
Stereo Processing FPGA	VP-M3A	Approach	A6
Addressed Aspects: VP-A2, VP-A3			
Description: Stereo processing is a computationally intensive process. As the visual odometry of the ARDEA system works on the depth data, this is also time-critical and is constantly required. ARDEA therefore uses an FPGA to calculate the depth data. SGM is used as the algorithm.			
Applies: G7			

F.4.6 ARDEA Stack IT

Model Name:	Model Kind:	Model Type:	ID:
ARDEA Stack IT	VP-M3I	Implementation	I3
Addressed Aspects: VP-A2, VP-A3			
Description: The ARDEA stack also contains all of the system's IT components. The high-level computer is implemented using an Intel NUK board, as this delivers high computing power in a small installation space. The low-level computer is implemented using a BeagleBone Black, which is compact, provides sufficient computing power and a native CAN interface to control the motor controllers. The FPGA is implemented by a XiLinx Spartan 6.			
Implements: A6, A5			

F.5 ARDEA Physical Power

F.5.1 Limited Power Resources

Model Name:	Model Kind:	Model Type:	ID:
Limited Power Resources	VP-M4G	Guideline	G8
Addressed Aspects: VP-A4			
Description: The available energy limits the possible applications of the system. The flight duration of a multicopter is always limited by the available energy. This therefore limits the possible uses of such a system.			

Rationale 21: ***Larger battery can lead to smaller operating radius***
The operating radius and therefore the possible applications cannot be increased arbitrarily by choosing the size of the battery. A larger battery results in more weight. This increases energy consumption in hovering flight, but also results in lower dynamics. This in turn means that a heavier battery can lead to a smaller operating radius despite its higher capacity, as the system consumes more energy and has to move more slowly.

Rationale 22: ***Weight saving leads directly to energy saving***
The biggest lever for saving energy is saving weight. Therefore, only the components that are actually needed should be carried.

F.5.2 Unlimited Power Resources

Model Name:	Model Kind:	Model Type:	ID:
Unlimited Power Resources (ARDEA)	VP-M4G	Guideline	G9
Addressed Aspects: VP-A4			
Description: Sufficient energy reserves are always available for the system. It is therefore not necessary to optimize the energy requirement.			

Identical to AIMM G8

Rationale 23: ***Energy consumption of the propulsion system is crucial***
By far the largest energy consumer in flight is the propulsion system. Optimization of the other components is therefore of secondary importance

F.5.3 Power Setup

Model Name:	Model Kind:	Model Type:	ID:
Power Setup	VP-M4A	Approach	A7
Addressed Aspects: VP-A4			
Description: The system uses the same energy source to power all components. During the flight, this is a battery. During the configuration of the system, this can be replaced by a power supply unit. The voltage of the central energy source is then transformed via various voltage converters to the voltages required by the components. To enable a battery change without restarting the computer components, two voltage sources can be connected simultaneously to ensure an uninterrupted power supply.			
Applies: G2, G8, G9			

F.5.4 ARDEA Stack Power

Model Name:	Model Kind:	Model Type:	ID:
ARDEA Stack Power	VP-M4I	Implementation	I4
Addressed Aspects: VP-A2, VP-A3			
Description: ARDEA's power management is integrated into the system stack. This enables operation even without the system frame.The standard flight battery is a LiPo battery with four cells. In order to prevent uncontrolled charging currents when changing from an empty to a full battery, an electronic circuit ensures that no energy can be fed back.			
Implements: A7			

F.6 ARDEA Physical Safety

F.6.1 Safety Operator

Model Name:	Model Kind:	Model Type:	ID:
Safety Operator (ARDEA)	VP-M5G	Guideline	G10
Addressed Aspects: VP-A5			
Description: A safety operator is responsible for the safety of the system and monitors the system during operation. This person must ensure that no persons are harmed and that neither the system nor its surroundings are damaged.			

Identical zu AIMM G9

F.6.2 Safety Pilot

Model Name:	Model Kind:	Model Type:	ID:
Safety Pilot	VP-M5A	Approach	A8
Addressed Aspects: VP-A5			
Description: For ARDEA, the safety of the system is ensured by the safety pilot. This pilot ensures that no person comes into contact with the system. If the system exhibits undesirable behavior, the safety pilot intervenes at the low level motion command level. The safety pilot can also trigger an emergency stop, i.e. a stop of all motors.			
Applies: G10, G11			

Similar to AIMM A6

Rationale 24: ***Emergency stop not always safe***
An emergency stop is not a safe option for a flight system as it causes the system to crash. This leads to major damage to the system. It can also endanger people in the vicinity. Depending on the situation, it may nevertheless be the best solution to trigger an emergency stop, especially if the system is no longer controllable.

Rationale 25: ***Qualification of a safety pilot***
The safety pilot must be able to control the system manually. The challenge with a drone,

unlike many other systems, is that control is necessary to ensure safe operation. The safety pilot must guarantee this and be able to take over the drone at any time and in any condition.

F.7 ARDEA Physical Interface

F.7.1 Low Level Motion Commands

Model Name:	Model Kind:	Model Type:	ID:
Low Level Motion Commands (ARDEA)	VP-M6G	Guideline	G11
Addressed Aspects: VP-A6			
Description: The *Robot* should also be able to be moved directly by the user. The ARDEA system therefore provides a low-level motion command interface that enables movements to be commanded directly.			

Identical to AIMM G11

F.7.2 Remote Controller

Model Name:	Model Kind:	Model Type:	ID:
Remote Controller (ARDEA)	VP-M6A	Approach	A9
Addressed Aspects: VP-A6			
Description: A remote control is used to control the movements. This enables the specification of speed setpoints for the *Robot* movement.			
Applies: G11			

Similar to AIMM A8

Rationale 26: ***All degrees of freedom of the system controllable***
The remote control for ARDEA enables full control of the system. The remote controller therefore has access to all possible actions of the system. This differs from the AIMM system, for example, where the manipulator cannot be controlled remotely.

F.7.3 ARDEA Remote Controller

Model Name:	Model Kind:	Model Type:	ID:
ARDEA Remote Controller	VP-M6I	Implementation	I5
Addressed Aspects: VP-A5, VP-A6			
Description: Remote control is via a standard Spectrum remote control for model airplanes. This product implements the full ARDEA remote control concept. This includes the low-level commands, but also additional channels, e.g. to switch motors on and off. The technical requirements such as range or size of the receiver also meet the ARDEA requirements.			
Implements: A8, A9			

Appendix G

LRU2 Mission View

This chapter describes the *Mission View* of the LRU2 system. It begins with an overview of all identified *Models* and their relationships with each other. The individual *Architecture Models* are then presented.

G.1 LRU2 Mission Overview

***Guideline Models*:**

- Low Level Tasks
- Continuous Abstraction Levels
- High Level Mission Tasks
- Task Sequence Mission
- Situation and Event based Mission Phases
- Collaborative Execution
- Autonomous Execution
- Online Mission Modification
- Knowledge gain about Environment
- Robot Mission Modification
- No robotic expert needed
- Scientist in the Loop

***Approach Models*:**

- Task Primitives
- Skill-driven Task (De)composition
- Skill Set and Behaviors
- Robotic Explorer
- Supervised Exploration
- Autonomous Exploration
- Local Autonomy
- Adaptive Tasks
- Mission Control for Scientist
- Task Specific GUIs

***Implementation Models*:**

- RAFCON Mission Task
- Operator as Information Source
- ROSMC Client
- RAFCON Execution
- ROSMC GUI
- Scientists GUIs

G.2 LRU2 Mission Abstraction

G.2.1 Low Level Tasks

Model Name:	Model Kind:	Model Type:	ID:
Low Level Tasks (LRU2)	VM-M1G	Guideline	G1
Addressed Aspects: VM-A1			
Description: A fundamental concept is to keep the level of abstraction of the *Tasks* as low as possible. The aim is to identify and explicitly name these smallest possible *Tasks*. These usually result from the decomposition of *Tasks* at higher levels of abstraction.			

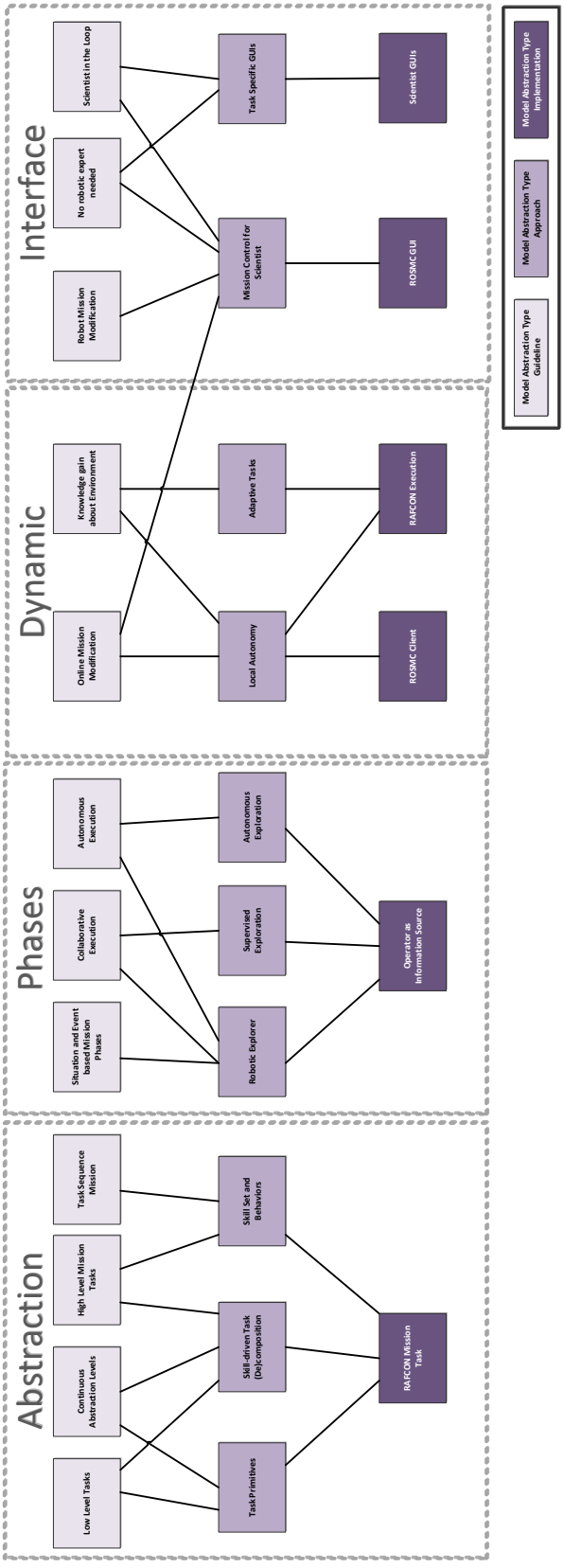

Figure G.1: *The Relations of the LRU2 Mission View Models*

Identical to AIMM G36

G.2.2 Continuous Abstraction Levels

Model Name:	Model Kind:	Model Type:	ID:
Continuous Abstraction Levels (LRU2)	VM-M1G	Guideline	G2
Addressed Aspects: VM-A1			
Description: The concept of Continuous Abstraction Levels is that *Tasks* are not assigned to a specific abstraction level. Instead, the *Task* abstraction levels of the *Tasks* form a continuous spectrum from low-level *Tasks* to very abstract high-level *Tasks* that define the *Mission*.			

Identical to AIMM G39

G.2.3 High Level Tasks

Model Name:	Model Kind:	Model Type:	ID:
High Level Tasks (LRU2)	VM-M1G	Guideline	G3
Addressed Aspects: VM-A1			
Description: High level *Tasks* describe *Tasks* at a very high level of abstraction. This makes it possible to specify what state the physical world should be in without specifying how this should be achieved. An example of a high-level *Task* is: “Tidy up the room”. Due to the high level of abstraction, high-level *Tasks* are usually very independent of the specific *Robot*. In addition, high-level *Tasks* are often also very independent of the current state of the physical world.			

Identical to AIMM G37

G.2.4 Task Sequence Mission

Model Name:	Model Kind:	Model Type:	ID:
Task Sequence Mission	VM-M1G	Guideline	G4
Addressed Aspects: VM-A1			
Description: A *Robot* often has various *Tasks* to fulfill in its *Mission*. There are often additional dependencies between these *Tasks* with regard to their sequence. A *Task* sequence makes it possible to define which *Tasks* must be executed in which order so that the *Mission* can be successfully completed.			

Rationale 1: ***Many missions can be represented as sequences***
Complex *Tasks* can often be split into individual, consecutive steps. Classic examples of this are assembly instructions for furniture or industrial manufacturing processes, for example. Generic instructions are also often given in the form of checklists. For example, after an accident, the accident site should first be secured, then first aid provided and then help called. Sequential *Missions* are simple, but still have a very wide range of application.

Rationale 2: ***Intuitive type of Mission design***
Task sequences are also popular with humans because they are easy to understand and implement. The status of a *Task* is very clear to understand and the successive actions are obvious without thinking, as the dependencies between the individual *Tasks* are very small. These characteristics also simplify the design of such a *Mission*.

Rationale 3: ***Compatible with many Task planners***
Many *Task* planners provide a sequence of *Subtasks* to be executed as a solution to a higher-level problem. A representation of the *Task* as a sequence enables the use of such approaches to *Task* planning.

Rationale 4: ***Unnecessary restrictions***
A disadvantage of the sequence is that the *Mission* is often restricted more than necessary. Some steps of the sequence may not be causally dependent on each other. For example, they could also be carried out in parallel or in a different order. The sequence is therefore often only one of several possible approaches to represent the *Mission*.

Rationale 5: ***Error handling only rudimentary possible***
A sequence can only handle errors in its individual steps to a very limited extent. Ultimately, the only way to handle errors is to repeat or skip individual steps. Otherwise, the *Mission* fails as soon as one of the steps fails.

Rationale 6: ***Reactive behavior cannot be represented***
Tasks that have to react to events during execution are difficult to represent in sequences. The simplification that the execution of a *Task* only depends on the execution of the previous *Task* excludes the possibility of *Tasks* being triggered by other dependencies.

G.2.5 Task Primitives

Model Name:	Model Kind:	Model Type:	ID:
Task Primitives (LRU2)	VM-M1A	Approach	A1
Addressed Aspects: VM-A1			
Description: *Task* primitives represent the lowest level of abstraction in the system. This means that a *Task* primitive does not contain any *Subtasks*. By applying the G2 concept, no fixed abstraction level is defined for *Task* primitives. *Task* primitives can therefore exist at different levels of abstraction. It is therefore also true that all *Tasks* can be decomposed into *Task* primitives.			
Applies: G1,G2			

Identical to AIMM A23

G.2.6 Skill-driven Task (De)composition

Model Name:	Model Kind:	Model Type:	ID:
Skill-driven Task (De)composition (LRU2)	VM-M1A	Approach	A2
Addressed Aspects: VM-A1			
Description: The system's *Skills* are used to decompose the *Tasks*. Each *Skill* can solve *Tasks*. If the *Skill* contains *Subskills*, these can also solve *Tasks*. The use of *Skills* therefore leads to *Task* decomposition. However, this often only happens at runtime, as the specific *Subtasks* are only determined based on the current situation. The structure of the *Skill* defines the abstraction levels of the *Tasks*.			
Applies: G1, G3, G2			

Identical to AIMM A24

G.2.7 Skill Set and Behaviors

Model Name:	Model Kind:	Model Type:	ID:
Skill Set and Behaviors	VM-M1A	Approach	A3
Addressed Aspects: VM-A1			
Description: The approach to defining *Missions* for LRU2 is based on G4. The possible *Tasks* are specified by high-level *Skills*. Any sequences can be created from this set of high-level *Tasks*. The use of G3 when creating the *Task* set ensures that the sequence can only be created on the basis of the *Mission* objectives. Technical dependencies are handled at the lower levels of abstraction. In addition, so-called behaviors can be defined for all *Tasks*. These specify the desired behavior of the system at runtime. For example, it may be desirable for the *Robot* to be permanently within communication range. If the *Robot* loses wireless contact, behaviors are automatically activated that attempt to re-establish wireless contact. Reactive behaviors can thus be specified within the list-based *Mission*.			
Applies: G3, G4			

Rationale 7: ***High Level Skills: A compact way of specifiing Tasks Sets***
According to G4, the *Mission* of LRU2 consists of a sequence of *Tasks*. In the realization, these *Tasks* cannot be chosen arbitrarily, but it is implicitly assumed that the *Tasks* are solvable for the system. Since *Tasks* are concrete by definition, an autonomous system can solve an infinite number of *Tasks*. If, for example, a *Robot* has the *Skill* to move to a position, then there are an infinite number of *Tasks* in continuous space that the *Robot* can solve. A very compact and practicable representation is therefore not the definition of the *Task* set but the use of *Skills* and their permissible parameter spaces. This also has the advantage that a solution approach automatically exists for each *Task*.

Rationale 8: ***Behaviors in list-based Mission***
Sequence-based *Missions* do not allow reactions to events to be specified directly. For example, a mobile system must move in such a way that communication with the system is possible. If this behavior is universal, it can be stored generally. In practice, however, there may be individual *Tasks* in which the rover has to explore a crater, for example. In this case, the behavior must be modified locally for a *Task*, e.g. exploration in the radio shadow is permitted. To make this possible, a set of behaviors is specified in addition to the *Task*, which are taken into account when performing the *Task*. However, more complex behavioral dependencies, e.g. combinations of several *Mission*, world and system states, cannot be represented in this way. In this case, the *Mission* would have to be extended to a behavior tree-based structure, for example.

G.2.8 RAFCON Mission Task

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Mission Task	VM-M1I	Implementation	I1
Addressed Aspects: VM-A1			
Description: The RAFCON framework is used to implement the various abstraction levels of the *Tasks*. The *Tasks* of all abstraction levels are represented here by the hierarchy concept. The high-level *Skills* that specify the *Mission Tasks* are each implemented by a separate state machine that can be executed at any time. These *Skills* must therefore first check the current status and then create a suitable context within the *Skill*. This enables the *Mission Tasks* to be sequenced as required.			
Implements: A1, A2, A3			

Rationale 9: *Context-independent Skills create high complexity*
Making *Skills* generally context-independent can create a high level of complexity depending on the *Task*. For some *Tasks*, e.g. driving to a position, context independence can be achieved relatively easily. This is because the system is able to move to positions regardless of the current configuration, environment, etc. Other *Tasks*, such as carrying out a measurement with an instrument, have many prerequisites that must be fulfilled. For example, the instrument must be docked -> the instrument must be available or retrieved -> there must be enough available capacity on the instrument holder -> another instrument must be put down –> etc. A precondition that the instrument needs to be on the rover would simplify the *Skill* enormously. However, this would then also have to be taken into account when sequencing the *Tasks*, which would increase the knowledge required by the *Mission* planner.

Rationale 10: *Hierarchy of Top Level Skills*
The high-level *Skills* set is on one level from a *Mission* perspective. All *Tasks* that can be solved with this set are possible elements of a *Mission*. From an implementation perspective, however, hierarchies between these high-level *Skills* are possible and reasonable. For example, the *Skill* Driving to a position is a high-level *Skill* that allows the operator to send the *Robot* to any position. However, this *Skill* is also required in many other high-level *Skills*, e.g. if the rover is to fetch something from the lander. At the implementation level, high-level *Skills* can also contain other high-level *Skills* as *Subskills*.

G.3 LRU2 Mission Phases

G.3.1 Situation and Event-based Mission Phases

Model Name:	Model Kind:	Model Type:	ID:
Situation and Eventbased Mission Phases	VM-M2G	Guideline	G5
Addressed Aspects: VM-A2			
Description: The individual *Mission* phases are not in a static sequence, but the current *Mission* phase changes due to situations and events. Situations are contexts that are deliberately created by the *Robot*. The *Robot* therefore plays an active role in such a phase transition. Events, on the other hand, are triggered by external circumstances, e.g. the environment, the operator or other agents. In this case, the *Robot* takes on a passive role. Both lead to a change in the current phase and thus to a change in the system behavior.			

G.3.2 Collaborative Execution

Model Name:	Model Kind:	Model Type:	ID:
Collaborative Execution	VM-M2G	Guideline	G6
Addressed Aspects: VM-A2			
Description: In the collaborative phase, interaction with another agent is required to solve the *Task*. This can be an operator or another *Robot*. A communication option is required for collaboration with an operator.			

Rationale 11: ***Collaborative Tasks/Autonomous Tasks***
The dependence on the operator can be artificial. For example, the *Robot* can select and collect a sample without the scientist. The *Task* is therefore technically solvable, but it is intended that the human makes the decision.

G.3.3 Autonomous Execution

Model Name:	Model Kind:	Model Type:	ID:
Autonomous Execution	VM-M2G	Guideline	G7
Addressed Aspects: VM-A2			
Description: In the autonomous execution phase, the system makes all decisions on its own. This applies to the *Mission* itself, but also to the reaction to unforeseen problems. However, the system can also decide to release control itself and thus initiate a phase change. In addition, every *Robot* has the option of aborting this phase from outside.			

G.3.4 Robotic Explorer

Model Name:	Model Kind:	Model Type:	ID:
Robotic Explorer	VM-M2A	Approach	A4
Addressed Aspects: VM-A2			
Description: The Robotic Explorer should relieve the operator as much as possible and simplify operation. Nevertheless, the operator must be able to make or monitor all important decisions. Additional challenges arise from the fact that the environment is only incompletely known and communication with the system may be limited or even impossible. The robotic system therefore alternates between two phases, Supervised Exploration, in which the human must make decisions, and Autonomous Exploration, in which the *Robot* makes all decisions independently. As shown in Figure G.2, the transition between the phases can be triggered by the *Robot* itself, by the operator or by events that occur.			
Applies: G5, G6, G7			

Rationale 12: ***Phase of the Robot does not match the phase of the operator***
The phases described correspond to the system phases. As far as possible, the operator will also monitor the autonomous phase of the *Robot*. From the operator's point of view, the *Robot* is therefore much more often in the monitored autonomy phase. However, this happens outside the *Robot*'s system boundaries and therefore has no influence on the *Robot* phases.

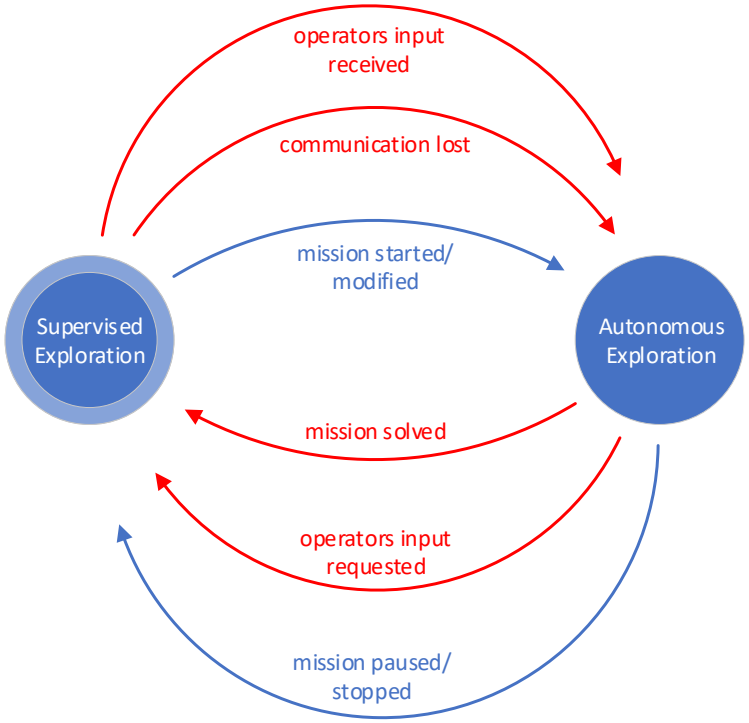

Figure G.2: *The phases of a robotic explorer and the transitions initiated by the Robot (red) and the environment (blue).*

G.3.5 Supervised Exploration

Model Name:	Model Kind:	Model Type:	ID:
Supervised Exploration	VM-M2A	Approach	A5

Addressed Aspects:
VM-A2

Description:
In the supervised exploration phase, the operator makes important decisions that determine the behavior or *Task* of the system. The aim of this phase is therefore to support these decisions from the *Robot*'s perspective. One challenge here is that many *Tasks* have to be performed by the system itself in order to master the technical complexity. For example, the scientist wants to select the sample, but accessibility, movement planning, object segmentation etc. remain the *Task* of the system. Another special characteristic is that the depth of intervention can take place at very different technical levels. For example, the operator determines which higher-level *Tasks* the system should perform. However, the scientist also determines exactly which sample is to be taken, including direct control of the pan-tilt unit to capture images of interesting formations.

Applies:
G6

Rationale 13: ***Self-activation of the autonomous phase***
A special feature of planetary exploration is that the *Robot* can also independently switch from the supervised exploration phase to the autonomous exploration phase. This is necessary because external events can interrupt communication, which practically requires full autonomy to either re-establish communication or, alternatively, to solve the *Mission* as far as possible autonomously.

G.3.6 Autonomous Exploration

Model Name:	Model Kind:	Model Type:	ID:
Autonomous Exploration	VM-M2A	Approach	A6
Addressed Aspects: VM-A2			
Description: During autonomous exploration, the *Robot* makes all decisions itself. In this phase, there are no commands from outside. The *Robot* has a *Mission* to fulfill, but must also react to events and protect itself. For example, the *Robot* must ensure that the battery charge level is sufficient. An important part of this phase is deciding when the *Robot* leaves the autonomous phase on its own. This can be a complex task as a communication link must be established.			
Applies: G7			

G.3.7 Operator as Information Source

Model Name:	Model Kind:	Model Type:	ID:
Operator as Information Source	VM-M2I	Implementation	I2
Addressed Aspects: VM-A2			
Description: The autonomy framework RAFCON is used to implement the various phases and their transitions. From the implementation point of view, the *Robot* is always in an autonomous state, i.e. RAFCON controls the *Robot*'s behavior at all times and in all phases. From the implementation perspective, the monitored autonomy phases are phases in which external information influences the behavior or information is requested from the operator. The operator is integrated into the sequence control like other sources of information.			
Implements: A4, A5, A6			

Rationale 14: ***Operator as a source of information***
The integration of the operator as a source of information from an implementation perspective has several advantages. For example, the operator can be easily integrated into different abstraction levels, as components serve as information sources at each level. The change between autonomous and monitored phases is therefore limited to the selection of the information source. If one source fails, it is possible to switch to the other.

G.4 LRU2 Mission Dynamic

G.4.1 Online Mission Modification

Model Name:	Model Kind:	Model Type:	ID:
Online Mission Modification	VM-M3G	Guideline	G8
Addressed Aspects: VM-A3			
Description: The full LRU2 *Mission* is designed in advance and then executed on the *Robot*. During execution, however, events may occur that make it necessary to modify the *Mission*. The reasons for such a change can be manifold. For example, exploration with a *Robot* may take longer than planned, making it necessary to redistribute the *Tasks*. However, it is also possible that the information gained shifts the focus of the exploration, which means, for example, that more samples have to be taken.			

Rationale 15: ***Static Mission with modifications***
A *Mission* with online modifications has similarities to an adaptive *Mission*. However, this is only the case if the operator is part of the system. From the operator's point of view, it is also an adaptive *Mission*. For the *Robot*, however, the *Mission* is static. The overall goal, e.g. geological exploration, is not part of the *Robot*'s *Mission*.

G.4.2 Knowledge gain about Environment

Model Name:	Model Kind:	Model Type:	ID:
Knowledge gain about Environment	VM-M3G	Guideline	G9
Addressed Aspects: VM-A3			
Description: During a space mission, it can be assumed that the environment is largely static. Particularly during the first exploration missions, there are no other actors on site who could cause changes. However, knowledge about the environment is incomplete and sometimes incorrect. Only during the mission further information can be collected, which may influence the *Mission* and thus require adaptive behavior.			

Rationale 16: ***Similar to AIMM G44***
Gaining new insights about the environment is fundamentally different from physically changing the environment AIMM G44. From the point of view of the *Robot*'s *Mission*,

however, it is of secondary importance whether a large boulder has moved or has always been in a different place. The *Mission* must be adapted accordingly as soon as new findings are available.

G.4.3 Local Autonomy

Model Name:	Model Kind:	Model Type:	ID:
Local Autonomy	VM-M3A	Approach	A7
Addressed Aspects: VM-A3			
Description: In order to be able to react to external *Mission* changes G8 as well as unforeseen events, the approach is to give the *Robot* the possibility to adapt the *Mission* itself. To do this, the *Robot* is given the full *Mission*, i.e. the sequence of all *Tasks* to be performed. If this is changed, the sequence of *Tasks* is updated. The *Robot* can also modify the *Task* sequence and report back to the *Mission* control accordingly.			
Applies: G8, G9			

Rationale 17: *Task executing agent*
An alternative approach is to keep the *Task* sequence in the *Mission* control outside the *Robot*. From there, the *Robot* would receive its next *Task* after completing a *Task*. This approach simplifies *Mission* modifications, as the *Robot* only ever knows its current *Task*. However, this also prevents the *Robot* from reacting autonomously to events, e.g. a sampling point cannot be reached. As long as the operator has contact with the system, the situation can be resolved. If communication is restricted, however, it becomes impossible.

Rationale 18: *Interleaved interface*
The *Mission* specification of the LRU2 is overlaid by local autonomy. This means that the individual *Tasks* and their sequence are already specified in the *Mission* from the outside, but the *Robot* has the option of adapting them. This combines the advantages of *Mission* planning from the outside, which is based on more information and resources, with *Mission* planning on the system, which enables higher autonomy and reliability.

Rationale 19: *Knowledge as basis of intelligent decision making*
Even if the *Robot* does not change the *Mission* itself, knowledge about it is the basis for intelligent decisions. This means that *Tasks* below the *Mission* level, such as recharging the batteries, can be scheduled at appropriate points.

G.4.4 Adaptive Tasks

Model Name:	Model Kind:	Model Type:	ID:
Adaptive Tasks (LRU2)	VM-M3A	Approach	A8
Addressed Aspects: VM-A3			
Description: Changes are to be expected at the lower levels of abstraction, as knowledge about the physical environment changes. For LRU2, adaptive *Tasks* are therefore used at the lower levels of abstraction. These make it possible to react flexibly to these changes. This also means that the *Tasks* on the lower abstraction levels are only defined at runtime.			
Applies: G9			

Identical to AIMM A30

G.4.5 RAFCON Execution

Model Name:	Model Kind:	Model Type:	ID:
RAFCON Execution (LRU2)	VM-M3I	Implementation	I3
Addressed Aspects: VM-A3			
Description: The *Mission* is executed using the RAFCON framework. This makes it possible to represent the static missions in the high-level area as *Task* state machines. The *Skill* concept, which is also implemented in RAFCON, is used to implement the adaptive *Tasks*. This decides which specific *Tasks* are executed based on the current situation.			
Implements: A7, A8			

Identical to AIMM I19

G.4.6 ROSMC Client

Model Name:	Model Kind:	Model Type:	ID:
ROSMC Client	VM-M3I	Implementation	I4
Addressed Aspects: VM-A3			
Description: The ROSMC client is also implemented in RAFCON. This top-level state machine implements the execution of the *Mission*. The *Task* list is not instantiated directly as a state machine, i.e. as a sequence of *Task* states, but the sequencing is implemented in a so-called decider state, which is linked to all high-level *Skills*. Based on the *Task* list and the outcome of the last *Task*, the next *Task* is determined and parameterized at runtime.			
Implements: A7			

Rationale 20: *Decider vs. Statemachine*
The use of a decider state has several advantages. The first advantage is that the state machine itself is independent of the *Mission*. A change to the *Mission* during runtime can also be applied without changing the state machine. Another advantage of the decider architecture is that even long missions remain very compact. The complexity is determined by the number of *Skills* and not by the *Mission* elements. In addition, the same checks must be carried out between *Tasks*, e.g. *Mission* paused. In a statemachine implementation, the same check state would always have to be introduced between each *Task* state. The disadvantage of the decider solution is that the current state of the *Mission* cannot be read from the state of the state machine. A start within the sequence is also not possible directly via RAFCON with the decider solution.

G.4.7 Robot Mission Modification

Model Name:	Model Kind:	Model Type:	ID:
Robot Mission Modification	VM-M4G	Guideline	G10
Addressed Aspects: VM-A4			
Description: The *Robot*'s *Mission* must be adaptable both in the planning phase and during execution. Therefore, an interface is needed that makes this possible. In order to be able to make adjustments during the *Mission*, it is also necessary to display the current status of the *Mission*, as parts of the *Mission* that have already been executed can no longer be changed.			

G.4.8 No robotic expert needed

Model Name:	Model Kind:	Model Type:	ID:
No robotic expert needed	VM-M4G	Guideline	G11
Addressed Aspects: VM-A4			
Description: The *Robot* should be operable by non-robotic experts. A general requirement for all user interfaces is therefore that no expert knowledge is required to use them.			

Rationale 21: ***System boundary Mission***
GUIs are generally used for high-level interfaces in particular. These programs often run on separate operating stations, e.g. in mission control center for the LRU2. Often several *Robots* can also be controlled via these programs. Nevertheless, the operating program is considered as part of the *Robot Mission* and not as part of the *Mission Environment*.

G.4.9 Scientist in the Loop

Model Name:	Model Kind:	Model Type:	ID:
Scientist in the Loop	VM-M4G	Guideline	G12
Addressed Aspects: VM-A4			
Description: An important requirement for the LRU2 interface is that the scientist can intervene at any time. From the interface's point of view, the *Robot* has an assistance function. It should therefore be possible for humans to make important decisions. These decisions can be made at very different levels of abstraction. For example, the scientist should be involved in the planning and execution of the overall *Mission*. However, they should also be able to select certain samples or carry out certain measurements with instruments.			

G.4.10 Mission Control for Scientist

Model Name:	Model Kind:	Model Type:	ID:
Mission Control for Scientist	VM-M4A	Approach	A9

Addressed Aspects:
VM-A4

Description:
A *Mission* control program is used for *Mission* planning and execution. This enables the scientist to plan, monitor and modify the *Robot Mission*. The accessible abstraction level here is the specification, distribution and sequencing of high-level *Tasks*. The specification of *Tasks* is supported by the *Mission* control program by offering the available *Skills* of the *Robot* and requesting the necessary parameters from the user. Once the *Skill* has been fully parameterized, the *Task* is defined and can be added to the *Task* list. The Mission Control program then makes it possible to modify the order of the *Tasks* and synchronize them with the *Robot*. The Mission Control GUI then allows the *Mission* to be started, paused or canceled. During execution, the status of the *Mission* is displayed.

Applies:
G10, G11, G12

G.4.11 Task Specific GUIs

Model Name:	Model Kind:	Model Type:	ID:
Task Specific GUIs	VM-M4A	Approach	A10
Addressed Aspects: VM-A4			
Description: Task-specific GUIs are used to integrate the scientist at the lower levels of abstraction. Here, decisions can also be made by the scientist below the high-level *Tasks*. As these GUIs are individually adapted to *Task* sets, it is possible to intervene at very deep levels. For example, the pan-tilt unit can be moved directly in such a GUI or certain filter wheels can be set to capture a spectral image.			
Applies: G11, G12			

G.4.12 ROSMC GUI

Model Name:	Model Kind:	Model Type:	ID:
ROSMC GUI	VM-M4I	Implementation	I5
Addressed Aspects: VM-A4			
Description: To design and customize the sequential *Mission*, the tool ROSMC [99] is used. It enables the user to intuitively create, execute and monitor multi-robot missions. It is based on *Task* sequences that can be synchronized with the various *Robots*. A *Task* is created by parameterizing a *Skill*. This parameterization can be done directly in an input mask in ROSMC. However, certain parameters can also be defined in a 3D viewer (positions, objects). As these two forms of input are synchronized, the operator can use the preferred method in each case. During execution, the status of the *Mission* is displayed using the list, but also in the 3D viewer using the current positions of the *Robots*.			
Implements: A9			

G.4.13 Scientist GUIs

Model Name:	Model Kind:	Model Type:	ID:
ROSMC	VM-M4I	Implementation	I6
Addressed Aspects: VM-A4			
Description: The Scientist GUI is used to integrate the scientist into the process during sampling. The GUI is triggered by the *Robot*. In this GUI, the scientist can select the desired sample. The *Robot* can also be moved locally and potential samples are highlighted using image segmentation. The corresponding sample is then selected by clicking in the image and the *Robot* picks it up autonomously and continues with the autonomous *Mission* execution.			
Implements: A10			

Glossary

4C acronym of Four Software Concerns according to Radestock and Eisenbach [96]. 44, 53, 66, 133, 134, 319

AD acronym of *Architecture Description*. 131, 133, 135

AD Element abbreviation of *Architecture Description Element* [59] 48, 99, 100, 107, 139

AFE acronym of *Architecture Framework Element*. 39, 47–49, 96, 97, 100

Approach *Model Abstraction Type* with intermediate abstraction level. This type is used to describe how the *Guidelines* are used and combined vi, 50–52, 57, 65, 80, 103–105, 108, 109, 117, 119, 121, 123, 125, 126, 128, 129, 131, 135, 140, 156, 180, 228, 256, 276, 294, 317

Architecture fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution [59] v, 23–29, 39, 40, 42, 45, 47, 49–54, 57, 64, 68, 78, 79, 92–94, 96, 97, 99–101, 103, 104, 111, 116, 117, 119, 121, 125, 126, 129, 132–134, 136, 139–144, 148, 151, 186, 189, 315–317

Architecture Context contains the documentation of the elements of an architectural description that do not contain architectural decisions but rather classify the environment of the architecture. This contains a brief description of the system-of-interest and its environment, the *Stakeholder* identification and system specific *Concerns* vi, ix, 47, 48, 97, 100–103, 111, 113, 117, 125, 147

Architecture Decision documentation of decisions regarding the *Architecture* [59] 49, 51, 55, 56, 64, 65, 78, 79, 92, 93, 97, 100, 101, 103–105, 107, 119, 125, 128, 133, 136, 139, 141, 142, 315

Architecture Description work product used to express an *Architecture* [59] v, ix, 13, 23, 26–30, 38, 39, 46, 47, 49–51, 53, 95, 96, 99–101, 103, 105, 111, 113, 115–117, 123, 125, 126, 128, 129, 131–137, 139–143, 315, 317

Architecture Description Element is any construct related to an *Architecture Description*. Every *Stakeholder*, *Concern*, *Architecture Viewpoint*, *Architecture View*, *Model Kind*, *Architecture Model*, *Architecture Decision* and *Rationale* is considered an AD element. [59] 39, 48, 99, 315

Architecture Description Process process of describing an *Architecture* v, vi, 100, 101, 103, 105, 112, 115

Architecture Framework conventions, principles and practices for the description of *Architectures* established within a specific domain of application and/or community of stakeholders [59] v, 13, 18, 23, 24, 26, 28–30, 38–40, 42, 44, 46–48, 51, 53, 94, 96, 134, 136, 137, 139, 317, 318

Architecture Framework Element all *Architecture* agnostic *AD Elements* defining the *Architecture Framework*. v, 39, 48, 96, 99, 100

Architecture Model an *Architecture View* is composed of one or more *Architecture Models*. An *Architecture Model* uses modelling conventions appropriate to the *Concerns* to be addressed. These conventions are specified by the *Model Kind* governing that *Model*. [59] 28, 47, 51, 52, 100, 105–107, 119, 135, 136, 155, 179, 227, 255, 275, 293, 315, 316, 318

Architecture Rationale Records explanation, justification or reasoning about architecture decisions that have been made. The rationale for a decision can include the basis for a decision, alternatives and trade-offs considered, potential consequences of the decision and citations to sources of additional information. [59] 107, 119, 136, 141, 318

Architecture View expresses the *Architecture* of the system-of-interest in accordance with an *Architecture Viewpoint* [59] 28, 103, 315, 316, 320

Architecture Viewpoint work product establishing the conventions for the construction, interpretation and use of *Architecture Views* to frame specific system *Concerns* [59] 27, 28, 39, 47, 50, 315, 316, 320

Aspect abbreviation of *Viewpoint Aspect*. 64, 65, 79, 93, 97, 107, 117, 119, 121, 123, 126, 140, 143

Autonomy *Robotic Concern* that deals with the approach that autonomous robotic system can achieve flexibility, usability and dependability by solving problems on their own. 43, 46, 67, 81, 116, 125, 126, 129, 319

Capability generation of specific information through calculations based on specific data to solve a specific problem. Definition 4.1 v, vi, ix, 41, 53, 58–63, 66–76, 78, 94, 96, 101, 119–121, 123, 125, 133, 134, 141, 179, 181, 184–190, 228, 231–233, 242, 246, 264, 316, 317, 319, 320

Capability trigger is a *Correspondence Rule* between *Viewpoints Capabilities* and *Skills* 95

Communication *Software Concern* that deals with the exchange of data, with a foundation of communication paradigms such as request-reply, synchronous and asynchronous. [96] 44, 46, 53, 58, 63, 64, 66, 319

Complexity *Robotic Concern* that frames the fact that complexity is one of the major challenges in robotic systems 35, 43, 45–47, 49, 52, 55, 58, 67, 81, 96, 126, 318, 319

Computation *Software Concern* is concerned with the data processing algorithms required by an application, with a foundation in traditional paradigms such as functional programming and object-oriented programming. [96] 44–46, 53, 58, 63, 64, 66, 319

Concern interest in a system relevant to one or more of its *Stakeholder* [59] v, ix, 27–29, 39, 41–49, 53, 55–58, 64, 65, 67, 70, 78, 79, 81, 96, 97, 100, 103, 117, 133, 134, 136, 142, 147, 149, 151–153, 315, 316, 318, 319

Configuration *Software Concern* that determines which system components should exist, and how they are inter-connected, and is based on principles of software architecture. [96] 44, 46, 53, 67, 70, 78, 319

Coordination *Software Concern* that is concerned with the interaction of the various system components. [96] 44, 46, 53, 67, 70, 78, 319

Correspondence are used to express *Architecture* relations of interest within an *Architecture Description* [59] 28, 39, 54, 94, 95, 139

Correspondence Rule are used to express *Architecture* relations of interest within an *Architecture Framework* [59] v, 28, 39, 94–97, 316, 317, 319

D&I abbreviation of *Decision & Interpretation* components 69, 73, 232

Decision & Interpretation *Skill* components that make decisions or interpretations based on data. 70, 72, 73, 317

Dependability *Robotic Concern* that is concerned with the robotic requirement that a *Robot* need to solve *Tasks* reliable 42, 43, 45, 46, 55, 57, 67, 78–81, 103, 319

Developer group of *Stakeholders* interested in the development of the *Robot* 40, 41, 43, 47, 53, 96, 103, 140, 149, 152, 319

Flexibility *Robotic Concern* that frames the robotic requirement to solve various *Tasks* 35, 42, 43, 45, 46, 55, 58, 67, 81, 92, 319

Guideline *Model Abstraction Type* with highest abstraction level. This type is used to describe general concepts and guidelines vi, 49–52, 57, 65, 80, 103–105, 107, 108, 117, 119, 121, 123, 125, 126, 128, 129, 131, 135, 140, 156, 180, 228, 256, 276, 294, 315

Hardware abstraction is a *Correspondence Rule* between *Viewpoints Physical* and *Capabilities* 95

Interface is a *Correspondence Rule* between *Viewpoints Physical* and *Mission* 95

Hardware Interface *Capability* type which is used to access the physical hardware components of a *Robot*. 61, 62

Implementation *Model Abstraction Type* with lowest abstraction level. This type is used to describe to describe concepts how *Approaches* are implemented. vi, 50–52, 57, 65, 80, 103–105, 109, 117, 119, 121, 123, 125, 126, 128, 131, 135, 140, 156, 180, 228, 256, 276, 294

Mission the collection of all *Tasks* a *Robot* has to solve vi, vii, ix, 31–33, 53, 66, 68, 80–83, 86–94, 96, 116, 123–128, 135, 136, 186, 192, 194, 195, 255, 257, 260, 264–266, 268, 269, 271–273, 293, 295–297, 299–302, 304, 305, 307–314, 317–320

Mission Control Environment *Mission Control Task* which is not part of the *Robot Mission* 83, 91, 318

Mission Control Interface connection between *Mission Control Robot* and *Mission Control Environment* 83, 88, 90–94, 272

Mission Control Robot *Mission Control Task* which is part of the *Robot Mission* 83, 91, 94, 318

Mission Control Task top level *Task* of a *Mission* 82, 83, 88, 91, 317, 318

Mission Environment the part of the *Mission* that is not solved by the *Robot* 33, 80–83, 86, 88, 91–93, 103, 136, 149, 265, 311

Model abbreviation of *Architecture Model* [59] vi, vii, 51, 52, 100, 103–110, 113, 117–131, 135, 136, 140, 141, 155–157, 179–181, 227–229, 255–257, 275–277, 293–295, 316, 318

Model Abstraction Type separation of *Concern Complexity* based on abstraction level of the architecture description v, 47–52, 57, 65, 66, 79, 80, 93, 94, 96, 97, 104–106, 113, 128, 136, 139–141, 315, 317

Model Kind conventions for a type of modelling [59] ix, 28, 47–49, 51–53, 55, 57, 58, 65–67, 79–81, 93, 94, 96, 100, 105, 106, 134, 139, 141, 143, 315, 316

Physical the physical part of the system. vi, vii, ix, 55, 58, 59, 71, 73, 96, 116–119, 129–131, 141, 155, 157, 275, 277, 317

Physical Environment the physical world without the *Robot*'s hardware 32, 57, 71, 103, 148–150

Process Skill *Skill Type* that has the purpose of encapsulating knowledge about a process. v, 74, 75

Rationale abbreviation of *Architecture Rationale* [59] 119, 121, 123, 125, 128, 131, 136, 315

Relations is describing relations between *Models* within a *View* vi, vii, 52, 103, 110, 117–124, 126, 127, 129, 130, 155, 157, 181, 229, 257, 277, 295

Resource allow the dependencies between *Skills* created by the physical world to be explicitly described. 69, 71–73, 79, 80

RoAF acronym of *Robot Architecture Framework*. v, vi, 7–9, 12, 18, 23, 24, 26, 28, 30, 32, 38, 39, 41–45, 47–52, 54, 56, 57, 59, 65, 68, 74, 79, 93–97, 99–101, 103–105, 107, 111–113, 115, 117, 125, 132–137, 139–145, 147, 149, 150, 319

Robot a machine which is designed to solve various *Tasks* in physical environments. 3.2 v, vi, 25, 26, 30–47, 49–59, 61–64, 66–68, 71, 73–78, 80–88, 90–96, 99–103, 111, 113, 115, 116, 125, 126, 129, 131–134, 139, 140, 142, 143, 147–152, 158, 159, 161, 162, 164, 165, 168, 172–178, 182–184, 186–189, 191, 192, 194–200, 202–210, 213, 217, 218, 222, 224, 228, 231–233, 237, 238, 240–243, 247, 249, 251–253, 258–260, 263–268, 279, 280, 282, 283, 290, 296, 297, 299–308, 310–314, 317–320

Robot Architecture Framework *Architecture Framework* for robotics domain 7, 13, 318

Robot Mission the part of the *Mission* that is solved by the *Robot* v, 32, 33, 53, 81, 82, 88, 91, 92, 311, 312, 317, 318

Robotic Concern the *Developer* five *Concerns* for the *Robot*. These are *Flexibility*, *Usability*, *Dependability*, *Complexity* and *Autonomy*. Together with the *Software Concerns*, these *Concerns* form the main *Concerns* specified in the *RoAF*. v, 42–45, 55, 81, 92, 96, 152, 316, 317, 319

Skill ability to solve a specific *Task* effectively by a combination of knowledge, capabilities and experience. Definition 4.2 v, vi, ix, 41, 46, 53, 67–80, 84, 85, 87, 94–96, 121–123, 125, 133, 134, 227–229, 231–236, 240–246, 248–252, 254, 262–264, 271, 298–300, 309, 310, 312, 313, 316–320

Skill Primitive *Skill Type* that has the purpose of linking a *Capability* directly to a *Task*. v, 73–75, 232, 262

Skill Type specialization of the general *Skill* for a specific purpose. 68, 73–80, 318, 319

Software Concern identify relevant aspects from the perspective of the *Robot* as a complex, distributed software system. Theses are the *4C*: *Computation*, *Communication*, *Configuration* and *Coordination* identified by Radestock and Eisenbach [96] 18, 44, 45, 55, 57, 66, 78, 81, 92, 96, 316, 317, 319

Stakeholder individual, team, organization, or classes thereof, having an interest in a system [59] ix, 26–29, 39–42, 44–49, 55, 58, 64, 67, 81, 96, 97, 100, 103, 117, 125, 140, 147, 149–152, 155, 158, 159, 315–317, 319

Subskill *Skill* used as component in another *Skill* 69–73, 78, 85, 228, 231, 233–235, 262, 298, 300

Subtask *Task* that is used to solve a higher-level *Task*. 72, 84–87, 89, 149, 163, 230, 233, 234, 261, 262, 264, 297, 298

Task a defined modification of the physical world or a determination of information about the physical world v, 31, 33–38, 40, 42, 43, 45, 66–70, 72–92, 94, 95, 103, 116, 125, 133, 144, 147–150, 152, 158, 163–168, 175, 182, 186, 189, 192, 194, 200, 202, 207, 208, 228, 230–232, 234, 237–242, 247, 248, 256, 258–268, 270–272, 281, 294, 296–301, 304, 307–310, 312, 313, 317–319

Task Abstraction Skill *Skill Type* that has the purpose to allow a simplified programming. v, 77, 78

Task Interface connection between two or more *Tasks* 83, 85–88, 91, 93, 272

Task Programming Skill *Skill Type* that has the purpose to allow a simplified programming. v, 76, 78

Task solving is a *Correspondence Rule* between *Viewpoints Skills* and *Mission* 95

Usability *Robotic Concern* that deals with the robotic requirement that a *Robot* need to be usable to solve *Tasks* 35, 42, 43, 45, 46, 55, 81, 92, 319

User group of *Stakeholders* interested in the application of the *Robot* 40–42, 46, 96, 103, 149, 152

View abbreviation of *Architecture View* [59] vi, vii, ix, 27, 51, 54, 83, 94, 100, 101, 103–105, 110, 111, 113, 116–132, 134–136, 139–141, 155, 157, 179, 181, 227, 229, 255, 257, 275, 277, 293, 295, 318

Viewpoint abbreviation of *Architecture Viewpoint* [59] 29, 42, 47–49, 52–55, 57, 58, 67, 68, 94, 96, 97, 100, 103–105, 111, 113, 117, 126, 132–134, 139, 140, 142, 143, 316, 317, 319, 320

Viewpoint Aspect identifies the relevant aspects of a certain *Viewpoint*. 48–50, 55–58, 64–67, 78–81, 92–95, 105–111, 117, 119, 121, 123, 126, 131, 132, 134, 141, 143

Viewpoint Capabilities perspective from which the *Capabilities* and their communication with each other are visible ix, 53, 57, 58, 64–66, 94, 100, 143

Viewpoint Mission perspective from which the *Mission* is visible ix, 53, 80–83, 92–94, 100, 123, 133

Viewpoint Physical Perspective from which the *Robot* is viewed as a physical system. ix, 53, 55–57, 94, 95, 100, 101, 117, 129, 133, 135

Viewpoint Skills perspective in which *Skills* and their relationship are visible. ix, 53, 66–68, 70, 78–80, 94, 100, 121

Bibliography

[1] Kai Adam, Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. "Engineering Robotics Software Architectures with Exchangeable Model Transformations". In: *2017 First IEEE International Conference on Robotic Computing (IRC)*. Apr. 2017, pp. 172–179. DOI: 10.1109/IRC.2017.16.

[2] Aakash Ahmad and Muhammad Ali Babar. "Software architectures for robotic systems: A systematic mapping study". In: *Journal of Systems and Software* 122 (Dec. 2016), pp. 16–39. ISSN: 01641212. DOI: 10.1016/j.jss.2016.08.039.

[3] Rachid Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. "An Architecture for Autonomy". In: *The International Journal of Robotics Research* 17.4 (Apr. 1998), pp. 315–337. ISSN: 0278-3649. DOI: 10.1177/027836499801700402.

[4] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[5] Paul Backes, Kyle Edelberg, Peter Vieira, Won Kim, Alex Brinkman, Sawyer Brooks, Sisir Karumanchi, Gene Merewether, and Wyatt Ubellacker. "The intelligent robotics system architecture applied to robotics testbeds and research platforms". In: *2018 IEEE Aerospace Conference*. Vol. 2018-March. IEEE Computer Society, June 2018, pp. 1–8. ISBN: 9781538620144. DOI: 10.1109/aero.2018.8396770.

[6] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoğlu, and Georg Bartels. "Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents". In: Brisbane, QLD, Australia. Brisbane, QLD, Australia: IEEE, 2018, pp. 512–519. ISBN: 978-1-5386-3082-2. DOI: 10.1109/ICRA.2018.8460964.

[7] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. “CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Taipei, Taiwan, Oct. 2010, pp. 1012–1017.

[8] Rainer Bischoff, Tim Guhl, Erwin Prassler, Walter Nowak, Gerhard Kraetzschmar, Herman Bruyninckx, Peter Soetens, Martin Haegele, Andreas Pott, Peter Breedveld, Jan Broenink, Davide Brugali, and Nicola Tomatis. “BRICS - Best practice in robotics”. In: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. IEEE, 2010, pp. 1–8.

[9] Wout Boerdijk, Maximilian Durner, Martin Sundermeyer, and Rudolph Triebel. “Towards Robust Perception of Unknown Objects in the Wild”. In: *ICRA 2022 workshop on “Robotic Perception and Mapping: Emerging Techniques”*. 2022.

[10] Wout Boerdijk, Marcus G. Müller, Maximilian Durner, and Rudolph Triebel. “ReSyRIS - A Real-Synthetic Rock Instance Segmentation Dataset for Training and Benchmarking”. In: *2023 IEEE Aerospace Conference*. 2023, pp. 1–9. DOI: 10.1109/AERO55745.2023.10115802.

[11] Wout Boerdijk, Marcus Gerhard Müller, Maximilian Durner, Martin Sundermeyer, Werner Friedl, Abel Gawel, Wolfgang Stürzl, Zoltan-Csaba Marton, Roland Siegwart, and Rudolph Triebel. “Rock Instance Segmentation from Synthetic Images for Planetary Exploration Missions”. In: *2021 IEE/RSJ International Conference on Intelligent Robots and Systems, IROS (Workshops)*. Oktober 2021.

[12] Simon Bogh, Casper Schou, Thomas Ruehr, Yevgen Kogan, Andreas Dömel, Manuel Brucker, Christof Eberst, Riccardo Tornese, Christoph Sprunk, Gian Diego Tipaldi, and Trine Hennessy. “Integration and Assessment of Multiple Mobile Manipulators in a Real-World Industrial Production Facility”. In: *ISR/Robotik 2014; 41st International Symposium on Robotics*. June 2014, pp. 1–8.

[13] Peter Bonasso, D Kortenkamp, D Miller, and M Slack. “Experiences with an Architecture for Intelligent Reactive Agents”. In: *Intelligent Agents II: Agent Theories, Architectures, and Languages* 9.2-3 (2-3 1995), pp. 237–256. ISSN: 0952-813X. DOI: 10.1080/095281397147103.

[14] Boston Dynamics. *Atlas Gets a Grip | Boston Dynamics*. YouTube. Jan. 2023.

[15] Christoph Brand, Martin J. Schuster, Heiko Hirschmüller, and Michael Suppa. "Stereo-vision based obstacle mapping for indoor/outdoor SLAM". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 1846–1853. DOI: 10.1109/IROS.2014.6942805.

[16] Christoph Brand, Martin J. Schuster, Heiko Hirschmüller, and Michael Suppa. "Submap matching for stereo-vision based indoor/outdoor SLAM". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 5670–5677. DOI: 10.1109/IROS.2015.7354182.

[17] R Brooks. "A Robust Layered Control System for a Mobile Robot". In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. ISSN: 2374-8710. DOI: 10.1109/jra.1986.1087032. arXiv: 1010.0034.

[18] Rodney A. Brooks. "Intelligence without Reason". In: *Artificial Intelligence* 47.1-3 (1991), pp. 139–159. ISSN: 00220663. DOI: 10.4324/9781351001885-2.

[19] Rodney A. Brooks. "Intelligence without representation". In: *Artificial Intelligence* 47.1-3 (Jan. 1991), pp. 139–159. ISSN: 00043702. DOI: 10.1016/0004-3702(91)90053-M.

[20] Manuel Brucker, Maximilian Durner, Zoltán-Csaba Márton, Ferenc Bálint-Benczédi, Martin Sundermeyer, and Rudolph Triebel. "6DoF Pose Estimation for Industrial Manipulation Based on Synthetic Data". In: *Proceedings of the 2018 International Symposium on Experimental Robotics*. Ed. by Jing Xiao, Torsten Kröger, and Oussama Khatib. Cham: Springer International Publishing, 2020, pp. 675–684. ISBN: 978-3-030-33950-0.

[21] Manuel Brucker, Simon Léonard, Tim Bodenmüller, and Gregory D Hager. "Sequential scene parsing using range and intensity information". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 5417–5424. DOI: 10.1109/ICRA.2012.6224978.

[22] Davide Brugali, Gregory S. Broten, Antonio Cisternino, Diego Colombo, Jannik Fritsch, Brian Gerkey, Gerhard Kraetzschmar, Richard Vaughan, and Hans Utz. "Trends in robotic software frameworks". In: *Springer Tracts in Advanced Robotics* (2007). ISSN: 16107438. DOI: 10.1007/978-3-540-68951-5_15.

[23] Sebastian G. Brunner, Franz Steinmetz, Rico Belder, and Andreas Dömel. "RAFCON: A graphical tool for engineering complex, robotic tasks". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2016-November. Institute of Electrical and Electronics Engineers Inc.,

Oct. 2016, pp. 3283–3290. ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759506.

[24] Sebastian Georg Brunner, Andreas Dömel, Peter Lehner, Michael Beetz, and Freek Stulp. "Autonomous Parallelization of Resource-Aware Robotic Task Nodes". In: *IEEE Robotics and Automation Letters* 4.3 (3 July 2019), pp. 2599–2606. ISSN: 23773766. DOI: 10.1109/LRA.2019.2894463.

[25] H. Bruyninckx. "Open robot control software: the OROCOS project". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 3. May 2001, 2523–2528 vol.3. DOI: 10.1109/ROBOT.2001.933002.

[26] Herman Bruyninckx, Nico Hochgeschwender, Luca Gherardi, Markus Klotzbücher, Gerhard Kraetzschmar, Davide Brugali, A Shakhimardanov, J Paulus, M Reckhaus, H Garcia, D Faconti, and Peter Soetens. "The BRICS Component Model: A Model-Based Development Paradigm for Complex Robotics Software Systems". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, New York, USA: ACM Press, Mar. 2013, pp. 1758–1764. ISBN: 978-1-4503-1656-9. DOI: 10.1145/2480362.2480693.

[27] Kristin Bussmann, Lukas Meyer, Florian Steidle, and Armin Wedler. "Slip Modeling and Estimation for a Planetary Exploration Rover: Experimental Results from Mt. Etna". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2449–2456. DOI: 10.1109/IROS.2018.8594294.

[28] S. Caselli, F. Monica, and M. Reggiani. "YARA: A Software Framework Enhancing Service Robot Dependability". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 1970–1976. ISBN: 0-7803-8914-X. DOI: 10.1109/ROBOT.2005.1570402.

[29] J.H. Connell. "SSS: a hybrid architecture applied to robot navigation". In: *Proceedings 1992 IEEE International Conference on Robotics and Automation*. 1992, pp. 2719–2724. ISBN: 0-8186-2720-4. DOI: 10.1109/ROBOT.1992.219995.

[30] E. Coste-Maniere and R. Simmons. "Architecture, the backbone of robotic systems". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 67–72 vol.1. DOI: 10.1109/ROBOT.2000.844041.

[31] Steve Cousins. “Welcome to ROS Topics [ROS Topics]”. In: *IEEE Robotics & Automation Magazine* 17.1 (Mar. 2010), pp. 13–14. ISSN: 1558-223X. DOI: 10.1109/MRA.2010.935808.

[32] Steve Cousins, Brian Gerkey, Ken Conley, and Willow Garage. “Sharing Software with ROS [ROS Topics]”. In: *IEEE Robotics & Automation Magazine* 17.2 (June 2010), pp. 12–14. ISSN: 1558-223X. DOI: 10.1109/MRA.2010.936956.

[33] Kourosh Darvish, Enrico Simetti, Fulvio Mastrogiovanni, and Giuseppe Casalino. “A Hierarchical Architecture for Human–Robot Cooperation Processes”. In: *IEEE Transactions on Robotics* 37.2 (Apr. 2021), pp. 567–586. ISSN: 1941-0468. DOI: 10.1109/TRO.2020.3033715.

[34] Andreas Dömel, Simon Kriegel, Manuel Brucker, and Michael Suppa. “Autonomous pick and place operations in industrial production”. In: *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. Oct. 2015, pp. 356–356. DOI: 10.1109/URAI.2015.7358978.

[35] Andreas Dömel, Simon Kriegel, Michael Kaßecker, Manuel Brucker, Tim Bodenmüller, and Michael Suppa. “Toward fully autonomous mobile manipulation for industrial environments”. In: *International Journal of Advanced Robotic Systems* 14.4 (2017), p. 1729881417718588. DOI: 10.1177/1729881417718588. eprint: https://doi.org/10.1177/1729881417718588.

[36] Maximilian Durner, Wout Boerdijk, Yunis Fanger, Ryo Sakagami, David Lennart Risch, Rudolph Triebel, and Armin Wedler. “Autonomous Rock Instance Segmentation for Extra-Terrestrial Robotic Missions”. In: *2023 IEEE Aerospace Conference*. Mar. 2023, pp. 01–14. DOI: 10.1109/AERO55745.2023.10115717.

[37] Maximilian Durner, Wout Boerdijk, Martin Sundermeyer, Werner Friedl, Zoltán-Csaba Márton, and Rudolph Triebel. “Unknown Object Segmentation from Stereo Images”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 4823–4830. DOI: 10.1109/IROS51168.2021.9636281.

[38] Maximilian Durner, Simon Kriegel, Sebastian Riedel, Manuel Brucker, Zoltán-Csaba Márton, Ferenc Bálint-Benczédi, and Rudolph Triebel. “Experience-based optimization of robotic perception”. In: *2017 18th International Conference on Advanced Robotics (ICAR)*. 2017, pp. 32–39. DOI: 10.1109/ICAR.2017.8023493.

[39] Maximilian Durner, Zoltan-Csaba Marton, Simon Kriegel, Manuel Brucker, Sebastian Riedel, Dominik Meinzer, and Rudolph Triebel. "Automated Benchmarks and Optimization of Perception Tasks". In: *IROS 2017: 2nd Workshop on Machine Learning Methods for High-Level Cognitive Capabilities in Robotics*. Aug. 2017.

[40] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martín-Martín, Arne Sieverling, Vincent Wall, and Oliver Brock. "Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 4831–4835. DOI: 10.24963/ijcai.2017/676.

[41] Figure. *Figure Status Update - AI Trained Coffee Demo*. YouTube. Jan. 2024.

[42] Richard E. Fikes and Nils J Nilsson. "Strips: A new approach to the application of theorem proving to problem solving". In: *Artificial Intelligence* 2 (3-4 1971), pp. 189–208. ISSN: 00043702. DOI: 10.1016/0004-3702(71)90010-5.

[43] R James Firby and Marc G Slack. "Task Execution : Interfacing Networks to Reactive Skill Networks". In: *AAAI Spring Symposium*. 1995.

[44] Andre Fonseca Prince, Bernhard Vodermayer, Benedikt Pleintinger, Alexander Kolb, Emanuel Staudinger, Enrico Dietz, Susanne Schröder, Sven Frohmann, Fabian Seel, and Armin Wedler. "Design and Implementation of a Modular Mechatronics Infrastructure for Robotic Planetary Exploration Assets". In: *Proceedings of the International Astronautical Congress, IAC*. Oktober 2021.

[45] Erann Gat. "Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots". In: *Aaai* (1992), pp. 809–815. ISSN: 01635719. DOI: 10.1145/122344.122357.

[46] Erann Gat. "On Three-Layer Architectures". In: *Artificial intelligence and mobile robots* (1997), pp. 195–210. DOI: 10.1.1.165.5283.

[47] Erann Gat. "Reliable goal-directed reactive control of autonomous mobile robots". UMI Order No. GAX91-23728. PhD thesis. USA, 1991.

[48] Daniele Gianni, Niklas Lindman, Joachim Fuchs, and Robert Suzic. "Introducing the European Space Agency Architectural Framework for Space-Based Systems of Systems Engineering". In: *Complex Systems Design & Management*. Ed. by Omar Hammami, Daniel Krob, and Jean-Luc Voirin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 335–346. ISBN: 978-3-642-25203-7.

[49] Riccardo Giubilato, Cedric Le Gentil, Mallikarjuna Vayugundla, Martin Schuster, Teresa Vidal-Calleja, and Rudolph Triebel. "GPGM-SLAM: a Robust SLAM System for Unstructured Planetary Environments with Gaussian Process Gradient Maps". In: *Field Robotics* 2 (Aug. 2022), pp. 1721–1753.

[50] Riccardo Giubilato, Mallikarjuna Vayugundla, Martin J. Schuster, Wolfgang Stürzl, Armin Wedler, Rudolph Triebel, and Stefano Debei. "Relocalization With Submaps: Multi-Session Mapping for Planetary Rovers Equipped With Stereo Cameras". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 580–587. DOI: 10.1109/LRA.2020.2964157.

[51] Riccardo Giubilato, Mallikarjuna Vayugundla, Wolfgang Stürzl, Martin J. Schuster, Armin Wedler, and Rudolph Triebel. "Multi-Modal Loop Closing in Unstructured Planetary Environments with Visually Enriched Submaps". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 8758–8765. DOI: 10.1109/IROS51168.2021.9635915.

[52] Stevan Harnad. "The symbol grounding problem". In: *Physica D: Nonlinear Phenomena* 42.1-3 (June 1990), pp. 335–346. ISSN: 0167-2789. DOI: 10.1016/0167-2789(90)90087-6. arXiv: 9906002 [arXiv:cs.AI].

[53] R. Hartley and F. Pipitone. "Experiments with the subsumption architecture". In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. 1991, 1652–1658 vol.2. DOI: 10.1109/ROBOT.1991.131856.

[54] T. Hasegawa, T. Suehiro, and K. Takase. "A model-based manipulation system with skill-based execution". In: *IEEE Transactions on Robotics and Automation* 8.5 (1992), pp. 535–544. ISSN: 1042296X. DOI: 10.1109/70.163779.

[55] Matthias Hellerer, Martin J. Schuster, and Roy Lichtenheldt. "Software-in-the-Loop Simulation of a Planetary Rover". In: *The International Symposium on Artificial Intelligence, Robotics and Automation in Space*. June 2016.

[56] U. Hillenbrand, B. Brunner, C. Borst, and G. Hirzinger. "The Robutler: a Vision-Controlled Hand-Arm System for Manipulating Bottles and Glasses". In: *CD*. LIDO-Berichtsjahr=2004, 2004.

[57] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Pearson Education, Limited, 1999, p. 432. ISBN: 9780321643346.

[58] IEEE. "IEEE Standard for an Architectural Framework for the Internet of Things (IoT)". In: *IEEE Std 2413-2019* (2020), pp. 1–269. DOI: 10.1109/IEEESTD.2020.9032420.

[59] ISO. “ISO/IEC/IEEE 42010:2011 - Systems and software engineering – Architecture description”. In: *ISOIECIEEE 420102011E Revision of ISOIEC 420102007 and IEEE Std 14712000* 2011.March (2011), pp. 1–46. DOI: 10.1109/IEEESTD.2011.6129467.

[60] Sisir Karumanchi, Kyle Edelberg, Ian Baldwin, Jeremy Nash, Jason Reid, Charles Bergh, John Leichty, Kalind Carpenter, Matthew Shekels, Matthew Gildner, David Newill-Smith, Jason Carlton, John Koehler, Tatyana Dobreva, Matthew Frost, Paul Hebert, James Borders, Jeremy Ma, Bertrand Douillard, Paul Backes, Brett Kennedy, Brian Satzinger, Chelsea Lau, Katie Byl, Krishna Shankar, and Joel Burdick. “Team RoboSimian: Semi-autonomous Mobile Manipulation at the 2015 DARPA Robotics Challenge Finals”. In: *Journal of Field Robotics* 34.2 (2017), pp. 305–332. DOI: https://doi.org/10.1002/rob.21676. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21676.

[61] Gayane Kazhoyan, Simon Stelter, Franklin Kenghagho Kenfack, Sebastian Koralewski, and Michael Beetz. “The Robot Household Marathon Experiment”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 9382–9388. DOI: 10.1109/ICRA48506.2021.9560774.

[62] David Kortenkamp, Reid Simmons, and Davide Brugali. “Robotic Systems Architectures and Programming”. In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. 2nd ed. Cham: Springer International Publishing, 2016, pp. 283–306. ISBN: 978-3-319-32552-1. DOI: 10.1007/978-3-319-32552-1_12.

[63] Simon Kriegel, Manuel Brucker, Zoltan-Csaba Marton, Tim Bodenmüller, and Michael Suppa. “Combining object modeling and recognition for active scene exploration”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 2384–2391. DOI: 10.1109/IROS.2013.6696691.

[64] Simon Kriegel, Christian Rink, Tim Bodenmüller, and Michael Suppa. “Efficient next-best-scan planning for autonomous 3D surface reconstruction of unknown objects”. In: *Journal of Real-Time Image Processing* 10.4 (2015), pp. 611–631. ISSN: 1861-8219. DOI: 10.1007/s11554-013-0386-6.

[65] P.B. Kruchten. “The 4+1 View Model of architecture”. In: *IEEE Software* 12.6 (1995), pp. 42–50. DOI: 10.1109/52.469759.

[66] Hannah Lehner, Martin J. Schuster, Tim Bodenmüller, and Simon Kriegel. “Exploration with active loop closing: A trade-off between exploration efficiency and map quality”. In: *2017 IEEE/RSJ International Conference on*

Intelligent Robots and Systems (IROS). 2017, pp. 6191–6198. DOI: 10.1109/IROS.2017.8206521.

[67] Peter Lehner and Alin Albu-Schäffer. "Repetition sampling for efficiently planning similar constrained manipulation tasks". In: Vancouver, BC, Canada. Vancouver, BC, Canada: IEEE, 2017, pp. 2851–2856. ISBN: 978-1-5386-2683-2. DOI: 10.1109/IROS.2017.8206116.

[68] Peter Lehner and Alin Albu-Schäffer. "The Repetition Roadmap for Repetitive Constrained Motion Planning". In: *IEEE Robotics and Automation Letters* 3.4 (4 2018), pp. 3884–3891. ISSN: 2377-3774. DOI: 10.1109/LRA.2018.2856925.

[69] Peter Lehner, Sebastian Brunner, Andreas Dömel, Heinrich Gmeiner, Sebastian Riedel, Bernhard Vodermayer, and Armin Wedler. "Mobile manipulation for planetary exploration". In: *2018 IEEE Aerospace Conference*. Vol. 2018-March. IEEE Computer Society, Mar. 2018, pp. 1–11. ISBN: 9781538620144. DOI: 10.1109/AERO.2018.8396726.

[70] Peter Lehner, Máximo A. Roa, and Alin Albu-Schäffer. "Kinematic Transfer Learning of Sampling Distributions for Manipulator Motion Planning". In: Philadelphia, PA, USA. Philadelphia, PA, USA: IEEE, 2022, pp. 7211–7217. ISBN: 978-1-7281-9682-4. DOI: 10.1109/ICRA46639.2022.9811915.

[71] Peter Lehner, Ryo Sakagami, Wout Boerdijk, Andreas Dömel, Maximilian Durner, Giacomo Franchini, Andre Prince, Kristin Lakatos, David Lennart Risch, Lukas Meyer, Bernhard Vodermayer, Enrico Dietz, Sven Frohmann, Fabian Seel, Susanne Schröder, Heinz-Wilhelm Hübers, Alin Albu-Schäffer, and Armin Wedler. "Mobile Manipulation of a Laser-induced Breakdown Spectrometer for Planetary Exploration". In: *2023 IEEE Aerospace Conference*. Mar. 2023, pp. 1–19. DOI: 10.1109/AERO55745.2023.10115597.

[72] Daniel Leidner, Wissam Bejjani, Alin Albu-Schäffer, and Michael Beetz. "Robotic Agents Representing, Reasoning, and Executing Wiping Tasks for Daily Household Chores". In: *AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS*. 2016.

[73] Dilip Kumar Limbu, Yeow Kee Tan, Ridong Jiang, and Tran Ang Dung. "A software architecture framework for service robots". In: *2011 IEEE International Conference on Robotics and Biomimetics, ROBIO 2011*. 2011, pp. 1736–1741. ISBN: 9781457721373. DOI: 10.1109/ROBIO.2011.6181540.

[74] Matthias Lutz, Dennis Stampfer, Alex Lotz, and Christian Schlegel. "Service robot control architectures for flexible and robust real-world task execution: best practices and patterns". In: *Informatik 2014*. Bonn: Gesellschaft für Informatik e.V., 2014, pp. 1295–1306. ISBN: 978-3-88579-626-8.

[75] Philipp Lutz, Marcus Gerhard Müller, Moritz Maier, Samantha Stoneman, Teodor Tomic, Ingo von Bargen, Martin J. Schuster, Florian Steidle, Armin Wedler, Wolfgang Stürzl, and Rudolph Triebel. "ARDEA — An MAV with skills for future planetary missions". In: *Journal of Field Robotics* (Mai 2019).

[76] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074.

[77] Ivano Malavolta, Grace Lewis, Bradley Schmerl, Patricia Lago, and David Garlan. "How do you Architect your Robots? State of the Practice and Guidelines for ROS-based Systems". In: *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Oct. 2020, pp. 31–40.

[78] Juan Martínez-Moritz, Ismael Rodríguez, Korbinian Nottensteiner, Jean-Pascal Lutze, Peter Lehner, and Máximo A. Roa. "Hybrid Planning System for In-Space Robotic Assembly of Telescopes using Segmented Mirror Tiles". In: *2021 IEEE Aerospace Conference (50100)*. Big Sky, MT, USA. Big Sky, MT, USA: IEEE, 2021, pp. 1–16. ISBN: 978-1-7281-7437-2. DOI: 10.1109/AERO50100.2021.9438399.

[79] Luisa Mayershofer, Peter Lehner, Daniel Leidner, and Alin Albu-Schaeffer. "Task-Level Programming by Demonstration for Mobile Robotic Manipulators through Human Demonstrations based on Semantic Skill Recognition". In: *ISR Europe 2023; 56th International Symposium on Robotics*. 2023, pp. 22–29.

[80] Matthias Mayr, Francesco Rovida, and Volker Krueger. "SkiROS2: A Skill-Based Robot Control Platform for ROS". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 6273–6280.

[81] Valery Marcial Monthe, Laurent Nana, and Georges Edouard Kouamou. "A Model-Based Approach for Common Representation and Description of Robotics Software Architectures". In: *Applied Sciences* 12.6 (2022). ISSN: 2076-3417. DOI: 10.3390/app12062982.

[82] M. G. Müller, M. Durner, A. Gawel, W. Stürzl, R. Triebel, and R. Siegwart. "A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 9765–9772. DOI: 10.1109/IROS51168.2021.9636644.

[83] M. G. Müller, F. Steidle, M. J. Schuster, P. Lutz, M. Maier, S. Stoneman, T. Tomic, and W. Stürzl. "Robust Visual-Inertial State Estimation with Multiple Odometries and Efficient Mapping on an MAV with Ultra-Wide FOV Stereo Vision". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 3701–3708. DOI: 10.1109/IROS.2018.8594117.

[84] Marcus G Müller, Maximilian Durner, Wout Boerdijk, Hermann Blum, Abel Gawel, Wolfgang Stürzl, Roland Siegwart, and Rudolph Triebel. "Uncertainty Estimation for Planetary Robotic Terrain Segmentation". In: *2023 IEEE Aerospace Conference*. 2023, pp. 1–8. DOI: 10.1109/AERO55745.2023.10115611.

[85] Marcus G Müller, Samantha Stoneman, Ingo von Bargen, Florian Steidle, and Wolfgang Stürzl. "Efficient Terrain Following for a Micro Aerial Vehicle with Ultra-Wide Stereo Cameras". In: *2020 IEEE Aerospace Conference*. 2020, pp. 1–9. DOI: 10.1109/AERO47225.2020.9172781.

[86] Marcus Gerhard Müller, Jaeyoung Lim, Lukas Schmid, Hermann Blum, Wolfgang Stürzl, Abel Gawel, Roland Siegwart, and Müller Triebel Rudolph. "Interactive OAISYS: A photorealistic terrain simulation for robotics research". In: *ICRA 2022 Workshop on Releasing Robots into the Wild: Simulations, Benchmarks, and Deployment*. Mai 2022.

[87] Javier Muñoz, Peter Lehner, Luis E. Moreno, Alin Albu-Schäffer, and Máximo A. Roa. "CollisionGP: Gaussian Process-Based Collision Checking for Robot Motion Planning". In: *IEEE Robotics and Automation Letters* 8 (7 2023), pp. 4036–4043. ISSN: 2377-3774. DOI: 10.1109/LRA.2023.3280820.

[88] I.A.D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin. "CLARAty and challenges of developing interoperable robotic software". In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 3. Oct. 2003, 2428–2435 vol.3. DOI: 10.1109/IROS.2003.1249234.

[89] Nils J Nilsson. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[90] Nils J Nilsson. *Shakey the Robot*. Vol. 323. Sri International Menlo Park, California, 1984.

[91] Fabian Peller-Konrad, Rainer Kartmann, Christian R. G. Dreher, Andre Meixner, Fabian Reister, Markus Grotz, and Tamim Asfour. "A memory system of a robot cognitive architecture and its implementation in ArmarX". In: *Robotics and Autonomous Systems* 164 (2023), Art.–Nr.: 104415. ISSN: 0921-8890, 0167-8493, 1872-793X. DOI: 10.1016/j.robot.2023.104415.

[92] Luis A Pineda, Ivan Meza, Hector Aviles, Carlos Gershenson, Caleb Rascon, Monserrat Alvarado, and Lisset Salinas. "Ioca: Interaction-oriented cognitive architecture". In: *Research in Computer Science* 54 (2011), pp. 273–284.

[93] Luis A. Pineda, Arturo Rodríguez, Gibran Fuentes, Caleb Rascon, and Ivan V. Meza. "Concept and Functional Structure of a Service Robot". In: *International Journal of Advanced Robotic Systems* 12.2 (Jan. 2015), p. 6. ISSN: 1729-8814. DOI: 10.5772/60026. eprint: https://doi.org/10.5772/60026.

[94] Oliver Porges, Roberto Lampariello, Jordi Artigas, Armin Wedler, Christoph Borst, and Maximo A. Roa. "Reachability and Dexterity: Analysis and Applications for Space Robotics". In: *Workshop on Advanced Space Technologies for Robotics and Automation - ASTRA*. Mai 2015.

[95] En Yen Puang, Peter Lehner, Zoltan-Csaba Marton, Maximilian Durner, Rudolph Triebel, and Alin Albu-Schäffer. "Visual Repetition Sampling for Robot Manipulation Planning". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 9236–9242. DOI: 10.1109/ICRA.2019.8793942.

[96] Matthias Radestock and Susan Eisenbach. "Coordination in evolving systems". In: *Trends in Distributed Systems CORBA and Beyond* 1161 (1996), pp. 162–176. ISSN: 16113349. DOI: 10.1007/3-540-61842-2.

[97] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios S. Polydoros, Bjarne Großmann, Ronald P. A. Petrick, and Volker Krüger. "SkiROS—A Skill-Based Robot Control Platform on Top of ROS". In: 2017, pp. 121–160. DOI: 10.1007/978-3-319-54927-9_4.

[98] Francesco Rovida and Volker Kruger. "Design and development of a software architecture for autonomous mobile manipulators in industrial environments". In: *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, Mar. 2015, pp. 3288–3295. ISBN: 978-1-4799-7800-7. DOI: 10.1109/ICIT.2015.7125585.

[99] Ryo Sakagami, Sebastian G. Brunner, Andreas Dömel, Armin Wedler, and Freek Stulp. "ROSMC: A High-Level Mission Operation Framework for Heterogeneous Robotic Teams". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. May 2023, pp. 5473–5479. DOI: 10.1109/ICRA48891.2023.10161133.

[100] Ryo Sakagami, Andreas Dömel, Peter Lehner, Sebastian Riedel, Sebastian G. Brunner, Alin Albu-Schäffer, and Freek Stulp. "A Tree-Based World Model for Reducing System Complexity in Autonomous Mobile Manipulation". In:

IEEE Robotics and Automation Letters 9.7 (7 2024), pp. 6680–6687. ISSN: 2377-3774. DOI: 10.1109/LRA.2024.3410156.

[101] Ryo Sakagami, Florian S. Lay, Andreas Dömel, Martin J. Schuster, Alin Albu-Schäffer, and Freek Stulp. "Robotic world models—conceptualization, review, and engineering best practices". In: *Frontiers in Robotics and AI* 10 (2023). ISSN: 2296-9144. DOI: 10.3389/frobt.2023.1253049.

[102] Christian Schlegel, Alex Lotz, Matthias Lutz, and Dennis Stampfer. "Composition, Separation of Roles and Model-Driven Approaches as Enabler of a Robotics Software Ecosystem". In: *Software Engineering for Robotics*. Springer International Publishing, Dec. 2020, pp. 53–108. ISBN: 9783030664947. DOI: 10.1007/978-3-030-66494-7_3.

[103] Christian Schlegel, Andreas Steck, Davide Brugali, and Alois Knoll. "Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering". In: *Simulation, Modeling, and Programming for Autonomous Robots*. Ed. by Noriaki Ando, Stephen Balakirsky, Thomas Hemker, Monica Reggiani, and Oskar von Stryk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 324–335. ISBN: 978-3-642-17319-6.

[104] Korbinian Schmid, Heiko Hirschmüller, Andreas Dömel, Iris Grixa, Michael Suppa, and Gerd Hirzinger. "View Planning for Multi-View Stereo 3D Reconstruction Using an Autonomous Multicopter". In: *Journal of Intelligent & Robotic Systems* 65.1 (2012), pp. 309–323. ISSN: 1573-0409. DOI: 10.1007/s10846-011-9576-2.

[105] Korbinian Schmid, Philipp Lutz, Teodor Tomić, Elmar Mair, and Tomic Hirschmüller Heiko. "Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation". In: *Journal of Field Robotics* 31.4 (2014), pp. 537–570. DOI: https://doi.org/10.1002/rob.21506. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21506.

[106] Korbinian Schmid, Felix Ruess, and Darius Burschka. "Local reference filter for life-long vision aided inertial navigation". In: *17th International Conference on Information Fusion (FUSION)*. 2014, pp. 1–8.

[107] Korbinian Schmid, Felix Ruess, Michael Suppa, and Darius Burschka. "State estimation for highly dynamic flying systems using key frame odometry with varying time delays". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 2997–3004. DOI: 10.1109/IROS.2012.6385969.

[108] Korbinian Schmid, Michael Suppa, and Darius Burschka. “Towards Autonomous MAV Exploration in Cluttered Indoor and Outdoor Environments”. In: *RSS 2013 Workshop on Resource-Efficient Integration of Perception, Control and Navigation for Micro Air Vehicles (MAVs)* (2013).

[109] Korbinian Schmid, Teodor Tomic, Felix Ruess, Heiko Hirschmüller, and Michael Suppa. “Stereo vision based indoor/outdoor navigation for flying robots”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 3955–3962. DOI: 10.1109/IROS.2013.6696922.

[110] Florian Schmidt and Robert Burger. “How we deal with software complexity in robotics:‘Links and nodes’ and the ‘robotkernel’”. In: *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2014.

[111] Susanne Schröder, Fabian Seel, Enrico Dietz, Sven Frohmann, Peder Bagge Hansen, Peter Lehner, Andre Fonseca Prince, Ryo Sakagami, Bernhard Vodermayer, Armin Wedler, Anko Börner, and Heinz-Wilhelm Hübers. “A Laser-Induced Breakdown Spectroscopy (LIBS) Instrument for In-Situ Exploration with the DLR Lightweight Rover Unit (LRU)”. In: *Applied Sciences* 14.6 (2024). ISSN: 2076-3417. DOI: 10.3390/app14062467.

[112] Martin J. Schuster, Christoph Brand, Sebastian G. Brunner, Peter Lehner, Josef Reill, Sebastian Riedel, Tim Bodenmüller, Kristin Bussmann, Stefan Büttner, Andreas Dömel, Werner Friedl, Iris Grixa, Matthias Hellerer, Heiko Hirschmüller, Michael Kassecker, Zoltán-Csaba Márton, Christian Nissler, Felix Ruess, Michael Suppa, and Armin Wedler. “The LRU Rover for Autonomous Planetary Exploration and Its Success in the SpaceBotCamp Challenge”. In: *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. May 2016, pp. 7–14. ISBN: 9781509022557. DOI: 10.1109/ICARSC.2016.62.

[113] Martin J. Schuster, Christoph Brand, Heiko Hirschmüller, Michael Suppa, and Michael Beetz. “Multi-robot 6D graph SLAM connecting decoupled local reference filters”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 5093–5100. DOI: 10.1109/IROS.2015.7354094.

[114] Martin J. Schuster, Marcus G. Müller, Sebastian G. Brunner, Hannah Lehner, Peter Lehner, Andreas Dömel, Mallikarjuna Vayugundla, Florian Steidle, Philipp Lutz, Ryo Sakagami, Lukas Meyer, Rico Belder, Michal Smisek, Wolfgang Stürzl, Rudolph Triebel, and Armin Wedler. “Towards Heterogeneous Robotic Teams for Collaborative Scientific Sampling in Lunar and Planetary Environments”. In: *Workshop on Informed Scientific Sampling in Large-scale*

Outdoor Environments at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019). 2019.

[115] Martin J. Schuster, Marcus G. Müller, Sebastian G. Brunner, Hannah Lehner, Peter Lehner, Ryo Sakagami, Andreas Dömel, Lukas Meyer, Bernhard Vodermayer, Riccardo Giubilato, Mallikarjuna Vayugundla, Josef Reill, Florian Steidle, Ingo von Bargen, Kristin Bussmann, Rico Belder, Philipp Lutz, Wolfgang Stürzl, Michal Smíšek, Moritz Maier, Samantha Stoneman, Andre Fonseca Prince, Bernhard Rebele, Maximilian Durner, Emanuel Staudinger, Siwei Zhang, Robert Pöhlmann, Esther Bischoff, Christian Braun, Susanne Schröder, Enrico Dietz, Sven Frohmann, Anko Börner, Heinz-Wilhelm Hübers, Bernard Foing, Rudolph Triebel, Alin O. Albu-Schäffer, and Armin Wedler. "The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration". In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 5315–5322. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3007468.

[116] Martin J. Schuster, Korbinian Schmid, Christoph Brand, and Michael Beetz. "Distributed stereo vision-based 6D localization and mapping for multi-robot teams". In: *Journal of Field Robotics* 36.2 (2019), pp. 305–332. DOI: https://doi.org/10.1002/rob.21812. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21812.

[117] Marco Sewtz, Hannah Lehner, Yunis Fanger, Jan Eberle, Martin Wudenka, Marcus G. Müller, Tim Bodenmüller, and Martin J. Schuster. "URSim - A Versatile Robot Simulator for Extra-Terrestrial Exploration". In: *2022 IEEE Aerospace Conference (AERO)*. 2022, pp. 1–14. DOI: 10.1109/AERO53065.2022.9843576.

[118] R.G. Simmons. "Concurrent planning and execution for autonomous robots". In: *IEEE Control Systems Magazine* 12 (1 1992), pp. 46–50. ISSN: 1941-000X. DOI: 10.1109/37.120453.

[119] Emanuel Staudinger, Riccardo Giubilato, Martin J. Schuster, Robert Pöhlmann, Siwei Zhang, Andreas Dömel, Armin Wedler, and Armin Dammann. "Terrain-aware communication coverage prediction for cooperative networked robots in unstructured environments". In: *Acta Astronautica* 202 (2023), pp. 799–805. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2022.10.050.

[120] Emanuel Staudinger, Robert Pöhlmann, Siwei Zhang, Armin Dammann, Riccardo Giubilato, Ryo Sakagami, Peter Lehner, Martin J. Schuster, Andreas Dömel, Bernhard Vodermayer, Andre F. Prince, and Armin Wedler. "Enabling

Distributed Low Radio Frequency Arrays - Results of an Analog Campaign on Mt. Etna". In: *2023 IEEE Aerospace Conference*. Mar. 2023, pp. 1–12. DOI: 10.1109/AERO55745.2023.10115553.

[121] Franz Steinmetz and Roman Weitschat. "Skill parametrization approaches and skill architecture for human-robot interaction". In: *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. Vol. 2016-Novem. IEEE, Aug. 2016, pp. 280–285. ISBN: 9781509024094. DOI: 10.1109/COASE.2016.7743419.

[122] Andreas Stemmer and Simon Bøgh. *TAPAS Deliverable 3.1: Simplified Parameterisation of Assembly Skills*. Tech. rep. DLR, 2014, pp. 1–30.

[123] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. "Implicit 3D Orientation Learning for 6D Object Detection from RGB Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[124] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, and Rudolph Triebel. "Augmented Autoencoders: Implicit 3D Orientation Learning for 6D Object Detection". In: *International Journal of Computer Vision* 128.3 (2020), pp. 714–729. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01243-8.

[125] G. A. J. Sussman. "A Computational Model of Skill Acquisition". PhD thesis. 1973, pp. 359–395. ISBN: 9780521674102.

[126] Louis Touko Tcheumadjeu, Franz Andert, Qinrui Tang, Alexander Sohr, Robert Kaul, Jörg Belz, Philipp Lutz, Moritz Maier, Marcus G. Mliller, and Müller Stürzl Wolfgang. "Integration of an Automated Valet Parking Service into an Internet of Things Platform". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 662–668. DOI: 10.1109/ITSC.2018.8569230.

[127] Moritz Tenorth and Michael Beetz. "KNOWROB — knowledge processing for autonomous personal robots". In: St. Louis, MO, USA. St. Louis, MO, USA: IEEE, 2009, pp. 4261–4266. ISBN: 978-1-4244-3804-4. DOI: 10.1109/IROS.2009.5354602.

[128] Teodor Tomic and Sami Haddadin. "A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept. 2014. DOI: 10.1109/iros.2014.6943154.

[129] Teodor Tomic, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grixa, Felix Ruess, Michael Suppa, and Darius Burschka. "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue". In: *IEEE Robotics & Automation Magazine* 19.3 (Sept. 2012), pp. 46–56. ISSN: 1558-223X. DOI: 10.1109/MRA.2012.2206473.

[130] Teodor Tomić. "Evaluation of acceleration-based disturbance observation for multicopter control". In: *2014 European Control Conference (ECC)*. IEEE, 2014, pp. 2937–2944. DOI: 10.1109/ECC.2014.6862237.

[131] Teodor Tomić and Tomic Haddadin Sami. "Simultaneous estimation of aerodynamic and contact forces in flying robots: Applications to metric wind estimation and collision detection". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 5290–5296. DOI: 10.1109/ICRA.2015.7139937.

[132] Teodor Tomić, Philipp Lutz, Korbinian Schmid, Andrew Mathers, and Tomic Sami Haddadin. "Simultaneous contact and aerodynamic force estimation (s-CAFE) for aerial robots". In: *The International Journal of Robotics Research* 39.6 (2020), pp. 688–728. DOI: 10.1177/0278364920904788. eprint: https://doi.org/10.1177/0278364920904788.

[133] Teodor Tomić, Moritz Maier, and Tomic Haddadin Sami. "Learning quadrotor maneuvers from optimal control and generalizing in real-time". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 1747–1754. DOI: 10.1109/ICRA.2014.6907087.

[134] Teodor Tomić, Christian Ott, and Tomic Haddadin Sami. "External Wrench Estimation, Collision Detection, and Reflex Reaction for Flying Robots". In: *IEEE Transactions on Robotics* 33.6 (2017), pp. 1467–1482. DOI: 10.1109/TRO.2017.2750703.

[135] Teodor Tomić, Korbinian Schmid, Philipp Lutz, Andrew Mathers, and Tomic Haddadin Sami. "The flying anemometer: Unified estimation of wind velocity from aerodynamic power and wrenches". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 1637–1644. DOI: 10.1109/IROS.2016.7759264.

[136] Nikolaus Vahrenkamp, Mirko Wächter, Manfred Kröhnert, Kai Welke, and Tamim Asfour. "The Robot Software Framework ArmarX". In: *it - Information Technology* 57.2 (Mar. 2015), pp. 99–111. ISSN: 1611-2776. DOI: 10.1515/itit-2014-1066.

[137] Dominick Vanthienen, Markus Klotzbuecher, and Herman Bruyninckx. “The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming”. In: *JOSER: Journal of Software Engineering for Robotics* 5.1 (2014), pp. 17–35.

[138] Mallikarjuna Vayugundla, Florian Steidle, Michal Smisek, Martin J. Schuster, Kristin Bussmann, and Armin Wedler. “Datasets of Long Range Navigation Experiments in a Moon Analogue Environment on Mount Etna”. In: *ISR 2018; 50th International Symposium on Robotics*. 2018, pp. 1–7.

[139] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. “The CLARAty architecture for robotic autonomy”. In: *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*. Vol. 1. IEEE, 2001, pp. 1/121–1/132. ISBN: 0-7803-6599-2. DOI: 10.1109/AERO.2001.931701.

[140] Armin Wedler, Mathias Hellerer, Bernhard Rebele, Heiner Gmeiner, Bernhard Vodermayer, Tobias Bellmann, Stefan Barthelmes, Roland Rosta, Caroline Lange, Lars Witte, Nicole Schmitz, Martin Knapmeyer, Alexandra Czeluschke, Laurenz Thomsen, Christoph Waldmann, Sascha Flögel, Martina Wilde, and Yuto Takei. “ROBEX – COMPONENTS AND METHODS FOR THE PLANETARY EXPLORATION DEMONSTRATION MISSION”. In: *13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. ASTRA. ESAWebsite, 2015.

[141] Armin Wedler, Marcus Gerhard Müller, Martin Schuster, Maximilian Durner, Sebastian Brunner, Peter Lehner, Hannah Lehner, Andreas Dömel, Mallikarjuna Vayugundla, Florian Steidle, Ryo Sakagami, Lukas Meyer, Michal Smisek, Wolfgang Stürzl, Nicole Schmitz, Bernhard Vodermayer, Andre Fonseca Prince, Emanuel Staudinger, Matthias Hellerer, Roy Lichtenheldt, Enrico Dietz, Christian Braun, Bernhard Rebele, Wout Boerdijk, Riccardo Giubilato, Josef Reill, Moritz Kuhne, Jongseok Lee, Alejandro Fontan Villacampa, Ingo Bargen, Susanne Schröder, Sven Frohmann, Fabian Seel, Rudolph Triebel, Neal Yi-Sheng Lii, Esther Bischoff, Sean Kille, Kjetil Wormnes, Aaron Pereira, William Carey, Angelo P. Rossi, Laurenz Thomsen, Thorsten Graber, Thomas Krüger, Peter Kyr, Anko Börner, Kristin Bussmann, Gerhard Paar, Arnold Bauer, Stefan Völk, Andreas Kimpe, Heike Rauer, Heinz-Wilhelm Hübers, Johann Bals, Sören Hohmann, Tamim Asfour, Bernard Foing, and Alin Albu-Schäffer. “Preliminary Results for the Multi-Robot, Multi-Partner, Multi-Mission, Planetary Exploration Analogue Campaign on Mount Etna”. In: *Proceedings of the International Astronautical Congress, IAC*. Oktober 2021.

[142] Armin Wedler, Marcus Gerhard Müller, Martin Schuster, Maximilian Durner, Peter Lehner, Andreas Dömel, Florian Steidle, Mallikarjuna Vayugundla, Ryo Sakagami, Lukas Meyer, Michal Smisek, Wolfgang Stürzl, Nicole Schmitz, Bernhard Vodermayer, Andre Fonseca Prince, Anouk Ehreiser, Emanuel Staudinger, Robert Pöhlmann, Siwei Zhang, Matthias Hellerer, Roy Lichtenheldt, Dennis Franke, Antoine Francois Xavier Pignede, Walter Schindler, Manuel Schütt, Bernhard Rebele, Wout Boerdijk, Riccardo Giubilato, Josef Reill, Moritz Kuhne, Jongseok Lee, Susanne Schröder, Sven Frohmann, Fabian Seel, Enrico Dietz, Rudolph Triebel, Neal Yi-Sheng Lii, Esther Bischoff, Christian Braun, Sean Kille, Kjetil Wormnes, Aaron Pereira, William Carey, A.P. Rossi, Laurenz Thomsen, Thorsten Graber, Thomas Krüger, Anko Börner, Patrick Irmisch, Kristin Bussmann, Gerhard Paar, Arnold Bauer, Stefan Völk, H. Rauer, Heinz-Wilhelm Hübers, Johann Bals, Sören Hohmann, Tamim Asfour, B. Foing, and Alin Olimpiu Albu-Schäffer. "Finally! Insights into the ARCHES Lunar Planetary Exploration Analogue Campaign on Etna in summer 2022". In: *73rd International Astronautical Congress, IAC 2022*. Sept. 2022.

[143] Armin Wedler, Bernhard Rebele, Josef Reill, Michael Suppa, Heiko Hirschmüller, Christoph Brand, Martin Schuster, Bernhard Vodermayer, Heiner Gmeiner, Annika Maier, Bertram Willberg, Kristin Bussmann, Fabian Wappler, and Matthias Hellerer. "LRU – Lightweight Rover Unit". In: *Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. Mai 2015.

[144] Armin Wedler, Martin J. Schuster, Marcus G. Müller, Bernhard Vodermayer, Lukas Meyer, Riccardo Giubilato, Mallikarjuna Vayugundla, Michal Smisek, Andreas Dömel, Florian Steidle, Peter Lehner, Susanne Schröder, Emanuel Staudinger, Bernard Foing, and Josef Reill. "German Aerospace Center's advanced robotic technologyfor future lunar scientific missions". In: *Philosophical Transactions of the Royal Society A*. Discussion meeting issue 'Astronomy from the Moon: the next decades' organised and edited by Ian Crawford, Martin Elvis, Joseph Silk and John Zarnecki 379.2188 (Nov. 2020).

[145] Armin Wedler, Mallikarjuna Vayugundla, Hannah Lehner, Peter Lehner, Martin J. Schuster, Sebastian Georg Brunner, Wolfgang Stürzl, Andreas Dömel, Heinrich Gmeiner, Bernhard Vodermayer, Bernhard Rebele, Iris Lynne Grixa, Kristin Bussmann, Josef Reill, Bertram Willberg, Annika Maier, Peter Meusel, Florian Steidle, Michal Smisek, Matthias Hellerer, Martin Knapmeyer, Frank Sohl, Alexandra Heffels, Lars Witte, Caroline Lange, Roland Rosta, Norbert Toth, Stefan Völk, Andreas Kimpe, Peter Kyr, and Martina Wilde. "First Results of the ROBEX Analogue Mission Campaign: Robotic Deployment of Seismic

Networks for Future Lunar Missions". In: *68th International Astronautical Congress: Unlocking Imagination, Fostering Innovation and Strengthening Security, IAC 2017*. Vol. 68. 68th International Astronautical Congress (IAC). International Astronautical Federation (IAF), Sept. 2017.

[146] Armin Wedler, Martina Wilde, Andreas Dömel, Marcus Gerhard Müller, Josef Reill, Martin Schuster, Wolfgang Stürzl, Rudolph Triebel, Heinrich Gmeiner, Bernhard Vodermayer, Kristin Bussmann, Mallikarjuna Vayugundla, Sebastian Brunner, Hannah Lehner, Peter Lehner, Anko Börner, Rainer Krenn, Armin Dammann, Uwe-Carsten Fiebig, Emanuel Staudinger, Frank Wenzhöfer, Sascha Flögel, Stefan Sommer, Tamim Asfour, Michael Flad, Sören Hohmann, Martin Brandauer, and Alin Olimpiu Albu-Schäffer. "From single autonomous robots toward cooperative robotic interactions for future planetary exploration missions". In: *Proceedings of the International Astronautical Congress, IAC*. Preceedings of the 69th International Astronautical Congress (IAC). International Astronautical Federation (IAF), Oktober 2018.

[147] John A Zachman. "The zachman framework for enterprise architecture". In: *Primer for Enterprise Engineering and Manufacturing.[si]: Zachman International* (2003).

List of my prior Publications

[1] Andreas Dömel. *Entwicklung eines Pfadplaners für einen Virtual Reality-Versuchsstand*. Studienarbeit. Technische Universität München. 2007.

[2] Andreas Dömel. "Untersuchung von Samplingverfahren zur Erstellung aufgabenorientierter Bewegungskarten". Diplomarbeit. Technische Universität München, 2010.

[3] Korbinian Schmid, Heiko Hirschmüller, Andreas Dömel, Iris Grixa, Michael Suppa, and Gerd Hirzinger. "View Planning for Multi-View Stereo 3D Reconstruction Using an Autonomous Multicopter". In: *Journal of Intelligent & Robotic Systems* 65.1 (2012), pp. 309–323. ISSN: 1573-0409. DOI: 10.1007/s10846-011-9576-2.

[4] Teodor Tomic, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grixa, Felix Ruess, Michael Suppa, and Darius Burschka. "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue". In: *IEEE Robotics & Automation Magazine* 19.3 (Sept. 2012), pp. 46–56. ISSN: 1558-223X. DOI: 10.1109/MRA.2012.2206473.

[5] Simon Bogh, Casper Schou, Thomas Ruehr, Yevgen Kogan, Andreas Dömel, Manuel Brucker, Christof Eberst, Riccardo Tornese, Christoph Sprunk, Gian Diego Tipaldi, and Trine Hennessy. "Integration and Assessment of Multiple Mobile Manipulators in a Real-World Industrial Production Facility". In: *ISR/Robotik 2014; 41st International Symposium on Robotics*. June 2014, pp. 1–8.

[6] Andreas Dömel, Simon Kriegel, Manuel Brucker, and Michael Suppa. "Autonomous pick and place operations in industrial production". In: *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. Oct. 2015, pp. 356–356. DOI: 10.1109/URAI.2015.7358978.

[7] Sebastian G. Brunner, Franz Steinmetz, Rico Belder, and Andreas Dömel. "RAFCON: A graphical tool for engineering complex, robotic tasks". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2016-November. Institute of Electrical and Electronics Engineers Inc., Oct. 2016, pp. 3283–3290. ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759506.

[8] Sebastian Georg Brunner, Franz Steinmetz, Rico Belder, and Andreas Dömel. "RAFCON: a Graphical Tool for Task Programming and Mission Control". In: *RoboCup 2016: Robot World Cup XX*. Lecture Notes in Computer Science. Springer, 2016.

[9] Martin J. Schuster, Christoph Brand, Sebastian G. Brunner, Peter Lehner, Josef Reill, Sebastian Riedel, Tim Bodenmüller, Kristin Bussmann, Stefan Büttner, Andreas Dömel, Werner Friedl, Iris Grixa, Matthias Hellerer, Heiko Hirschmüller, Michael Kassecker, Zoltán-Csaba Márton, Christian Nissler, Felix Ruess, Michael Suppa, and Armin Wedler. "The LRU Rover for Autonomous Planetary Exploration and Its Success in the SpaceBotCamp Challenge". In: *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. May 2016, pp. 7–14. ISBN: 9781509022557. DOI: 10.1109/ICARSC.2016.62.

[10] Andreas Dömel, Simon Kriegel, Michael Kaßecker, Manuel Brucker, Tim Bodenmüller, and Michael Suppa. "Toward fully autonomous mobile manipulation for industrial environments". In: *International Journal of Advanced Robotic Systems* 14.4 (2017), p. 1729881417718588. DOI: 10.1177/1729881417718588. eprint: https://doi.org/10.1177/1729881417718588.

[11] Armin Wedler, Mallikarjuna Vayugundla, Hannah Lehner, Peter Lehner, Martin J. Schuster, Sebastian Georg Brunner, Wolfgang Stürzl, Andreas Dömel, Heinrich Gmeiner, Bernhard Vodermayer, Bernhard Rebele, Iris Lynne Grixa, Kristin Bussmann, Josef Reill, Bertram Willberg, Annika Maier, Peter Meusel, Florian Steidle, Michal Smisek, Matthias Hellerer, Martin Knapmeyer, Frank Sohl, Alexandra Heffels, Lars Witte, Caroline Lange, Roland Rosta, Norbert Toth, Stefan Völk, Andreas Kimpe, Peter Kyr, and Martina Wilde. "First Results of the ROBEX Analogue Mission Campaign: Robotic Deployment of Seismic Networks for Future Lunar Missions". In: *68th International Astronautical Congress: Unlocking Imagination, Fostering Innovation and Strengthening Security, IAC 2017*. Vol. 68. 68th International Astronautical Congress (IAC). International Astronautical Federation (IAF), Sept. 2017.

[12] Peter Lehner, Sebastian Brunner, Andreas Dömel, Heinrich Gmeiner, Sebastian Riedel, Bernhard Vodermayer, and Armin Wedler. “Mobile manipulation for planetary exploration”. In: *2018 IEEE Aerospace Conference*. Vol. 2018-March. IEEE Computer Society, Mar. 2018, pp. 1–11. ISBN: 9781538620144. DOI: 10.1109/AERO.2018.8396726.

[13] Armin Wedler, Martina Wilde, Andreas Dömel, Marcus Gerhard Müller, Josef Reill, Martin Schuster, Wolfgang Stürzl, Rudolph Triebel, Heinrich Gmeiner, Bernhard Vodermayer, Kristin Bussmann, Mallikarjuna Vayugundla, Sebastian Brunner, Hannah Lehner, Peter Lehner, Anko Börner, Rainer Krenn, Armin Dammann, Uwe-Carsten Fiebig, Emanuel Staudinger, Frank Wenzhöfer, Sascha Flögel, Stefan Sommer, Tamim Asfour, Michael Flad, Sören Hohmann, Martin Brandauer, and Alin Olimpiu Albu-Schäffer. “From single autonomous robots toward cooperative robotic interactions for future planetary exploration missions”. In: *Proceedings of the International Astronautical Congress, IAC*. Preceedings of the 69th International Astronautical Congress (IAC). International Astronautical Federation (IAF), Oktober 2018.

[14] Sebastian Georg Brunner, Andreas Dömel, Peter Lehner, Michael Beetz, and Freek Stulp. “Autonomous Parallelization of Resource-Aware Robotic Task Nodes”. In: *IEEE Robotics and Automation Letters* 4.3 (3 July 2019), pp. 2599–2606. ISSN: 23773766. DOI: 10.1109/LRA.2019.2894463.

[15] Martin J. Schuster, Marcus G. Müller, Sebastian G. Brunner, Hannah Lehner, Peter Lehner, Andreas Dömel, Mallikarjuna Vayugundla, Florian Steidle, Philipp Lutz, Ryo Sakagami, Lukas Meyer, Rico Belder, Michal Smisek, Wolfgang Stürzl, Rudolph Triebel, and Armin Wedler. “Towards Heterogeneous Robotic Teams for Collaborative Scientific Sampling in Lunar and Planetary Environments”. In: *Workshop on Informed Scientific Sampling in Large-scale Outdoor Environments at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019)*. 2019.

[16] Martin J. Schuster, Marcus G. Müller, Sebastian G. Brunner, Hannah Lehner, Peter Lehner, Ryo Sakagami, Andreas Dömel, Lukas Meyer, Bernhard Vodermayer, Riccardo Giubilato, Mallikarjuna Vayugundla, Josef Reill, Florian Steidle, Ingo von Bargen, Kristin Bussmann, Rico Belder, Philipp Lutz, Wolfgang Stürzl, Michal Smíšek, Moritz Maier, Samantha Stoneman, Andre Fonseca Prince, Bernhard Rebele, Maximilian Durner, Emanuel Staudinger, Siwei Zhang, Robert Pöhlmann, Esther Bischoff, Christian Braun, Susanne Schröder, Enrico Dietz, Sven Frohmann, Anko Börner, Heinz-Wilhelm Hübers, Bernard Foing, Rudolph Triebel, Alin O. Albu-Schäffer, and Armin Wedler. “The ARCHES Space-Analogue Demonstration Mission: Towards Heteroge-

neous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration". In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 5315–5322. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3007468.

[17] Armin Wedler, Martin J. Schuster, Marcus G. Müller, Bernhard Vodermayer, Lukas Meyer, Riccardo Giubilato, Mallikarjuna Vayugundla, Michal Smisek, Andreas Dömel, Florian Steidle, Peter Lehner, Susanne Schröder, Emanuel Staudinger, Bernard Foing, and Josef Reill. "German Aerospace Center's advanced robotic technologyfor future lunar scientific missions". In: *Philosophical Transactions of the Royal Society A*. Discussion meeting issue 'Astronomy from the Moon: the next decades' organised and edited by Ian Crawford, Martin Elvis, Joseph Silk and John Zarnecki 379.2188 (Nov. 2020).

[18] Armin Wedler, Marcus Gerhard Müller, Martin Schuster, Maximilian Durner, Sebastian Brunner, Peter Lehner, Hannah Lehner, Andreas Dömel, Mallikarjuna Vayugundla, Florian Steidle, Ryo Sakagami, Lukas Meyer, Michal Smisek, Wolfgang Stürzl, Nicole Schmitz, Bernhard Vodermayer, Andre Fonseca Prince, Emanuel Staudinger, Matthias Hellerer, Roy Lichtenheldt, Enrico Dietz, Christian Braun, Bernhard Rebele, Wout Boerdijk, Riccardo Giubilato, Josef Reill, Moritz Kuhne, Jongseok Lee, Alejandro Fontan Villacampa, Ingo Bargen, Susanne Schröder, Sven Frohmann, Fabian Seel, Rudolph Triebel, Neal Yi-Sheng Lii, Esther Bischoff, Sean Kille, Kjetil Wormnes, Aaron Pereira, William Carey, Angelo P. Rossi, Laurenz Thomsen, Thorsten Graber, Thomas Krüger, Peter Kyr, Anko Börner, Kristin Bussmann, Gerhard Paar, Arnold Bauer, Stefan Völk, Andreas Kimpe, Heike Rauer, Heinz-Wilhelm Hübers, Johann Bals, Sören Hohmann, Tamim Asfour, Bernard Foing, and Alin Albu-Schäffer. "Preliminary Results for the Multi-Robot, Multi-Partner, Multi-Mission, Planetary Exploration Analogue Campaign on Mount Etna". In: *Proceedings of the International Astronautical Congress, IAC*. Oktober 2021.

[19] Armin Wedler, Marcus Gerhard Müller, Martin Schuster, Maximilian Durner, Peter Lehner, Andreas Dömel, Florian Steidle, Mallikarjuna Vayugundla, Ryo Sakagami, Lukas Meyer, Michal Smisek, Wolfgang Stürzl, Nicole Schmitz, Bernhard Vodermayer, Andre Fonseca Prince, Anouk Ehreiser, Emanuel Staudinger, Robert Pöhlmann, Siwei Zhang, Matthias Hellerer, Roy Lichtenheldt, Dennis Franke, Antoine Francois Xavier Pignede, Walter Schindler, Manuel Schütt, Bernhard Rebele, Wout Boerdijk, Riccardo Giubilato, Josef Reill, Moritz Kuhne, Jongseok Lee, Susanne Schröder, Sven Frohmann, Fabian Seel, Enrico Dietz, Rudolph Triebel, Neal Yi-Sheng Lii, Esther Bischoff, Christian Braun, Sean Kille, Kjetil Wormnes, Aaron Pereira, William Carey, A.P. Rossi, Laurenz Thomsen, Thorsten Graber, Thomas Krüger, Anko Börner,

Patrick Irmisch, Kristin Bussmann, Gerhard Paar, Arnold Bauer, Stefan Völk, H. Rauer, Heinz-Wilhelm Hübers, Johann Bals, Sören Hohmann, Tamim Asfour, B. Foing, and Alin Olimpiu Albu-Schäffer. "Finally! Insights into the ARCHES Lunar Planetary Exploration Analogue Campaign on Etna in summer 2022". In: *73rd International Astronautical Congress, IAC 2022*. Sept. 2022.

[20] Peter Lehner, Ryo Sakagami, Wout Boerdijk, Andreas Dömel, Maximilian Durner, Giacomo Franchini, Andre Prince, Kristin Lakatos, David Lennart Risch, Lukas Meyer, Bernhard Vodermayer, Enrico Dietz, Sven Frohmann, Fabian Seel, Susanne Schröder, Heinz-Wilhelm Hübers, Alin Albu-Schäffer, and Armin Wedler. "Mobile Manipulation of a Laser-induced Breakdown Spectrometer for Planetary Exploration". In: *2023 IEEE Aerospace Conference*. Mar. 2023, pp. 1–19. DOI: 10.1109/AERO55745.2023.10115597.

[21] Ryo Sakagami, Sebastian G. Brunner, Andreas Dömel, Armin Wedler, and Freek Stulp. "ROSMC: A High-Level Mission Operation Framework for Heterogeneous Robotic Teams". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. May 2023, pp. 5473–5479. DOI: 10.1109/ICRA48891.2023.10161133.

[22] Ryo Sakagami, Florian S. Lay, Andreas Dömel, Martin J. Schuster, Alin Albu-Schäffer, and Freek Stulp. "Robotic world models—conceptualization, review, and engineering best practices". In: *Frontiers in Robotics and AI* 10 (2023). ISSN: 2296-9144. DOI: 10.3389/frobt.2023.1253049.

[23] Emanuel Staudinger, Riccardo Giubilato, Martin J. Schuster, Robert Pöhlmann, Siwei Zhang, Andreas Dömel, Armin Wedler, and Armin Dammann. "Terrain-aware communication coverage prediction for cooperative networked robots in unstructured environments". In: *Acta Astronautica* 202 (2023), pp. 799–805. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2022.10.050.

[24] Emanuel Staudinger, Robert Pöhlmann, Siwei Zhang, Armin Dammann, Riccardo Giubilato, Ryo Sakagami, Peter Lehner, Martin J. Schuster, Andreas Dömel, Bernhard Vodermayer, Andre F. Prince, and Armin Wedler. "Enabling Distributed Low Radio Frequency Arrays - Results of an Analog Campaign on Mt. Etna". In: *2023 IEEE Aerospace Conference*. Mar. 2023, pp. 1–12. DOI: 10.1109/AERO55745.2023.10115553.

[25] Samuel Bustamante, Ismael Rodríguez, Gabriel Quere, Peter Lehner, Maged Iskandar, Daniel Leidner, Andreas Dömel, Alin Albu-Schäffer, Jörn Vogel, and Freek Stulp. "Feasibility Checking and Constraint Refinement for Shared

Control in Assistive Robotics". In: *IEEE Robotics and Automation Letters* PP (99 2024), pp. 1–8. ISSN: 2377-3774. DOI: 10.1109/LRA.2024.3430710.

[26] Junsheng Ding, Ingmar Kessler, Alexander Perzylo, Markus Knauer, Andreas Dömel, Christoph Willibald, Sebastian Riedel, Stefan Profanter, Sebastian Brunner, Arsenii Dunaev, Le Li, and Manuel Brucker. "Intuitive Instruction of Robot Systems: Semantic Integration of Standardized Skill Interfaces". In: *IEEE International Conference on Industrial Informatics (INDIN)*. Aug. 2024.

[27] Martin Görner, Jennifer Cebulsky, Andreas Dömel, Maximilian Durner, Riccardo Giubilato, Moritz Kuhne, Marcus Gerhard Müller, Kristin Lakatos, Peter Lehner, Roy Lichtenheldt, Bernhard Rebele, Mattias Roser, Ryo Sakagami, Yunis Scheeler, Martin Schuster, Manuel Schütt, Wolfgang Stürzl, Mallikarjuna Vayugundla, and Armin Wedler. "The DLR Moon-Mars Test Site for Robotic Planetary Exploration". In: *International Conference on Space Robotics*. June 2024.

[28] Ryo Sakagami, Andreas Dömel, Peter Lehner, Sebastian Riedel, Sebastian G. Brunner, Alin Albu-Schäffer, and Freek Stulp. "A Tree-Based World Model for Reducing System Complexity in Autonomous Mobile Manipulation". In: *IEEE Robotics and Automation Letters* 9.7 (7 2024), pp. 6680–6687. ISSN: 2377-3774. DOI: 10.1109/LRA.2024.3410156.