Experiences Gained From Modeling and Solving Large Mapping Problems During System Design

Dr. Robert Hilbrich
German Aerospace Center
Rutherfordstr. 2
12489 Berlin, Germany
Email: robert.hilbrich@dlr.de

Dr. Michael Behrisch
German Aerospace Center
Rutherfordstr. 2
12489 Berlin, Germany
Email: michael.behrisch@dlr.de

***Abstract*—The rising complexity of embedded control systems and their increasing application to automate safety-critical or mission-critical tasks present a challenge for established development methodologies and tools. Are they able to handle the growing system complexity without compromising either system efficiency or its correctness? This challenge is addressed by the "correctness by construction" engineering principle. It aims to formalize error-prone and cumbersome engineering tasks to ensure correctness as well as efficiency despite high levels of complexity. A major obstacle in applying this principle in practice lies in the necessary formalization of "constructive tasks" for which human engineers with creative minds are still predominantly responsible.**

The authors applied this principle to "mapping problems", which occur during the design of several real-world embedded control systems. The tool suite ASSIST was developed to automate the "mapping process" and demonstrate the feasibility of this approach. It takes textual specifications of a mapping problem and its constraints as input from the systems engineer and uses Constraint Programming to synthesize valid and optimized solutions. In this contribution, the experiences gained from modeling and solving large-scale mapping problems as part of the design of embedded control systems are described in detail.

I. Introduction

A. Motivation and Background

Embedded control systems, such as *Attitude and Orbit Control Systems (AOCS)* in modern space vehicles, have seen a huge increase in complexity and capability over the last years. Today, astronauts are able to perform sophisticated maneuvers in space just by using a small joystick. The necessary adjustments of momentum wheels, reaction wheels, magnetic torques and thrusters to conduct the flight maneuver are automatically computed and entirely managed by computerized control systems. The complexity of maneuvering a vehicle in space is mostly hidden from the pilot. These computerized control systems are more and more used in different domains to automate safety-critical or mission-critical tasks in which small errors may have serious or even fatal consequences.

Due to the importance of these systems for the fulfillment of a mission objective, their design has to ensure *reliability* and *correctness* with no tolerance for undetected errors. At the same time, systems engineers have to exploit the potential of new and efficient – but also even more complex – hardware components. Multicore processors are only a recent example for the technological advancement of the basic hardware building blocks for these systems [1]. Their increased processing power is a necessary prerequisite to implement new features and capabilities. However, a growing functional complexity, increasing requirements and a rising complexity in the hardware and software architecture pose a significant challenge for current systems engineering methods and tools. Embedded control systems have reached a level of complexity in which their correct behavior can no longer be argued by simply observing their behavior. Due to the large amount of possible system states a *complete* a posteriori analysis for defects is no longer economically feasible [2], [3]. Therefore, alternatives approaches for the design of these systems must be explored.

A promising approach tries to argue the correctness of a system based on its *construction process* instead of simply observing its behavior after it has been built. This idea is called the "Correctness by Construction" principle. It has been applied to the design and implementation of several software projects [4], [5] and real-time systems [6], [7]. The authors of this paper extended the idea to the design of embedded control systems [8], [9].

Designing such a system requires the engineer to conduct a variety of cumbersome and error-prone tasks which affect the correctness of the entire system. From this set of engineering tasks, the synthesis of solutions for mapping problems has been selected by the authors as a starting to point to investigate the effectiveness of the "Correctness by Construction" principle.

B. Large-Scale Mapping Problems

Mapping problems in general refer to the assignment from elements of a set A to elements in another set B. This assignment has to satisfy additional constraints, which restrict the set of possible assignments. An assignment represents a solution to a mapping problem, if all elements of set A are assigned to an element of set B and all constraints are satisfied by this assignment.

For example, a common and very difficult mapping problem which occurs during the design of control systems is the assignment of software components (tasks or processes) to their hardware resources (processor cores, memory, ...). This assignment has to ensure, that the "resource demands" of the software components are fulfilled and that their timing and reliability requirements are satisfied.

Finding a correct solution for a mapping problem during system design is generally considered to be a difficult "constructive" task. In contrast to "analytical" tasks, it requires the creative mind of a human engineer who understands all design constraints in order to derive a valid solution within the vast design space. Constructive tasks are generally complex and difficult to automate. Therefore, they are still predominantly conducted by human engineers.

Unfortunately, finding a valid assignment in a mapping problem is often *not* enough. Usually, the assignment is also expected to be *optimized* – but not necessarily optimal – with respect to additional optimization criteria. The total weight of the system or the number of processors being used are typical examples for optimization goals during system design.

Constructing a correct and also optimal assignment *manually* requires a lot of effort and time. The systems engineer has to deal with the complexity of the *entire* system architecture while also having to satisfy many intricate safety requirements and taking optimization goals into consideration. Yet an incorrect solution may have *severe* effects on the correct behavior of the system and on the safety of the passengers. Therefore, exploring ways to *automate* the construction of optimized assignments for mapping problems despite the complexity of the system and without jeopardizing its correctness is an important research goal.

The authors successfully automated the construction process of solutions to several mapping problems during the development of *real-world* control systems. These systems contained several thousand mappable entities and tens of thousands of constraints, so that automated construction approaches have not been tried yet. The authors refer to this class of mapping problems as *large-scale mapping problems*. Solutions for this kind of mapping problems have been previously constructed manually by using spreadsheet calculators and an elaborated set of handcrafted macros and scripts. In the past, the construction of a single – and not necessarily optimized – assignment typically required 6 to 14 person months. This effort is significant – even with respect to the long product cycles in the aerospace domain. It reduces the agility of the engineering process and it does not support the engineer, who likes to explore the available design space.

II. Hypothetical Spacecraft Architecture

The characteristics for each control system and the specifics for each of the mapping problems differ significantly. Furthermore, they cannot be published without violating intellectual property rights. Therefore, the authors devised a *hypothetical spacecraft architecture* for this paper as an example to describe and illustrate the experiences gained from modeling and solving these large-scale mapping problems. This example is depicted in Figure 1.

As part of the development of this system, a mapping is required between a set of 3000 sensors and a set of 7000 ports on 50 distributed hubs. Each of these hubs acts as a gateway and data concentrator to the backbone network within the space vehicle. The space vehicle's *Attitude and Orbit Control System (AOCS)* is connected to the backbone network and requires reliable sensor data for its computations and the subsequent application of the thrusters.

Each sensor has to be connected with a dedicated cable to exactly one port on a hub. These ports cannot be chosen freely, because each sensor requires a port with a specific i/o type (analog, digital, . . .). All hubs in the system are equal with regard to their ports. Their design cannot be changed without a lengthy procurement process from external suppliers. The positions of the hubs within the spacecraft in Cartesian coordinates are known and fix.

Some of the ports on a hub are *internally connected*. For example, a group of *three* internally connected ports on a hub can be regarded as only *one* port with three differently shaped "sockets". (Differently shaped "sockets" are regarded as a different i/o types in this example.) Therefore, only one sensor can be connected to a group of internally connected ports, but the specific "socket" within this group can be chosen freely during the mapping process (see Figure 1).

In addition to these restrictions, the mapping has to satisfy *colocality* and *dislocality* requirements. A *colocality* is specified for a group of sensors and requires these sensors to be mapped to ports on the *same* hub – without requiring a particular hub. This requirement is a result of the desire to reduce the manufacturing costs. If a group of sensors are connected to the same hub, then prefabricated cables can be used to reduce the cabling effort. In the use-case for this paper, there are 125 colocality specifications in total. Each specification refers to a group of sensors comprising between four and 30 sensors.

Dislocality requirements on the other hand are not applied to reduce costs, but to increase reliability and fault tolerance of the control system. The failure of a hub may jeopardize the system safety, if vital sensor data is no longer available for the AOCS to properly actuate the thrusters. In order to prevent a hub becoming a single point of failure, different sensors providing similar sensor data must be connected to the backbone network via different hubs. Reliability considerations for the backbone network are beyond the scope of this paper.

Generally speaking, a dislocality can be specified in two different ways. In the basic case, a dislocality is specified for a single group sensors, which requires all of the sensors in the group to be mapped to different hubs. In the advanced case, a dislocality is specified for several *groups* of sensors. This requires all specified sensor groups to be mapped to different hubs, while the sensors within a group may be mapped to the same hub. In short, a hub, which is used by a sensor within one group, must not be used by any other sensor in another group.

The example in this paper contains 1 basic dislocality specification referring to 30 sensors and 45 advanced dislocality specifications. Each of the advanced dislocality specifications refers to between 3 and 8 sensor groups, whereas each of these groups contains between 2 and 900 sensors.

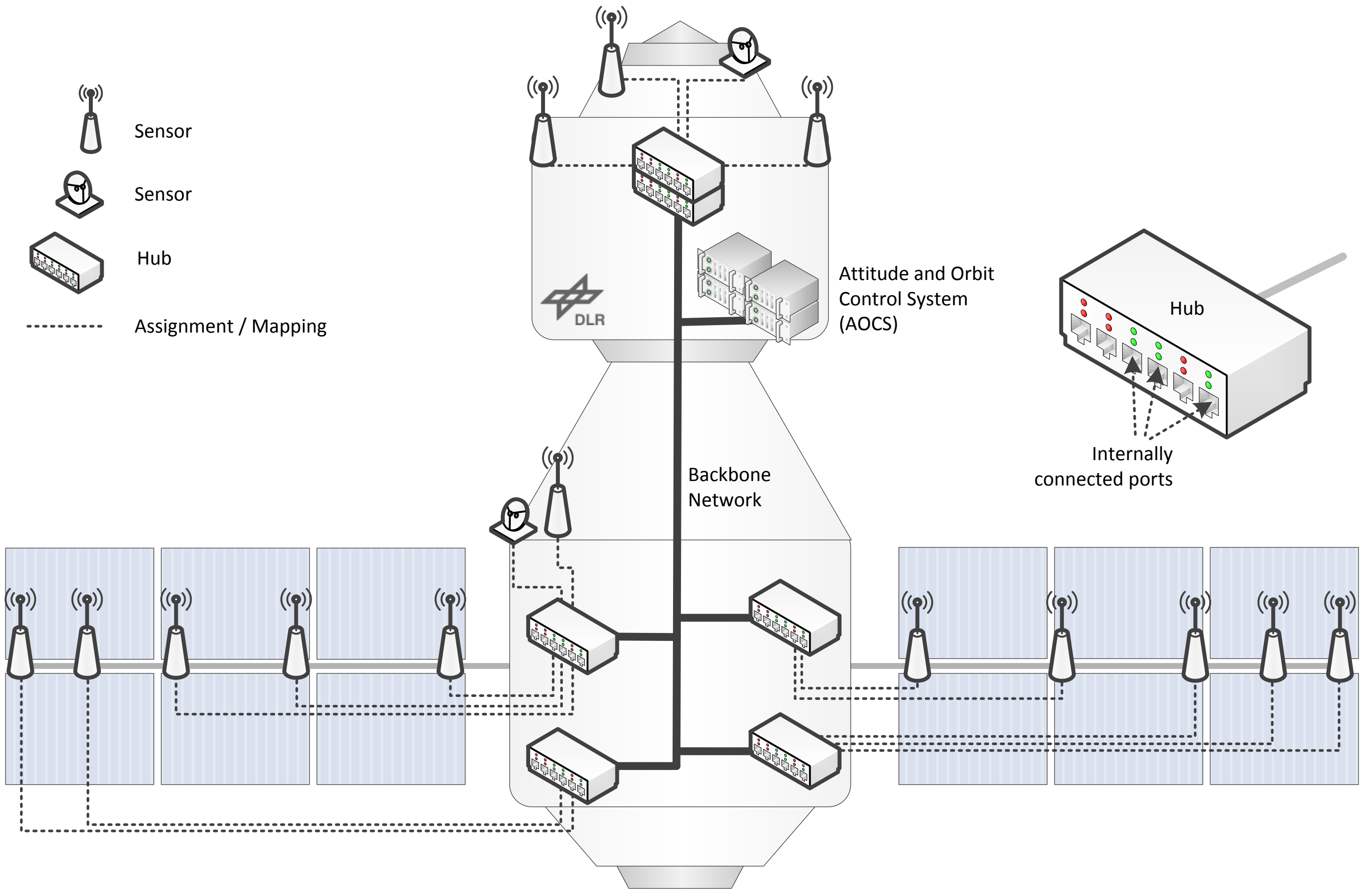

Sensor
Sensor
Hub
Assignment / Mapping
DLR
Attitude and Orbit Control System (AOCS)
Hub
Internally connected ports
Backbone Network

Fig. 1. Each sensor aboard a space vehicle must be connected to a single port on a sensor hub. Sensor data can then be transmitted via the backbone network to the Attitude and Orbit Control System (AOCS). Mapping sensors to ports must take the type of the port and several reliability constraints into account.

These intricate requirements significantly reduce the choice of ports for each sensor and illustrate the fact that manual construction approaches have reached the limit of their capability. The design space of valid, but not necessarily optimized solutions cannot be easily determined by manually examining the specification.

In addition to the specifications which define the correctness of a mapping solution, there is also an optimization goal to be pursued. For the example in this paper, the *minimization of the total cable weight* for the system is used. The total cable weight can be computed by multiplying the total length of all sensor cables with a specific cable weight factor. Calculating the length for a *single* sensor cable is done with a city block distance between the Cartesian coordinates for the sensor and its hub. In comparison to real systems, this goal is significantly simplified for illustration purposes. Additional effects, such as different weight factors for each i/o type or different electromagnetic shielding requirements for longer cables leading to higher cable weights, are not taken into account.

III. Automated Construction of Solutions

In order to achieve an automated construction of a mapping solution and to argue its correctness based on its synthesis, a formalization of the mapping problem and the synthesis steps for a solution are required. For smaller mapping problems, this has been successfully achieved based on Linear Integer Programming [10], [11], SMT-based solvers [12] or evolutionary algorithms [13]. However, these approaches reach their limits when large-scale mapping problems with limited gradient information to guide a search process are considered. The authors instead chose to transform a mapping problem into a semantically equivalent *Constraint Satisfaction Problem (CSP)* [14]–[16] and solve this CSP with *Constraint Programming* techniques [17], [18]. The advantages of using Constraint Programming in comparison to other techniques lie in the availability of powerful modeling elements, such as an allDifferent constraint, and the ease with which custom search heuristics can be implemented.

A. Constraint Satisfaction Problems

Constraint Programming refers to a set of techniques in artificial intelligence and operations research. These techniques assist in finding solutions for problems based on variables, which are affected by constraints. Each constraint defines valid or invalid solutions for a subset of these variables. In this paper, a subclass of constraint satisfaction problems is used to express mapping problems: *finite domain integer constraint* *satisfaction problems* in which each variable has a finite integer domain. Solutions for this problem class can be obtained by applying a combination of *search* techniques – including backtracking – and constraint *propagation* techniques for value elimination.

To illustrate the modeling approach of Constraint Satisfaction Problems, consider the well-known *Map Coloring* problem as an example. This problem asks, whether it is possible to color a map with only four colors in such a way, that neighboring countries have different colors. It can be formulated as a CSP by assigning an integer variable x_i for each country with the index i. The domain of each variable corresponds to the four colors: $D_{x_i} = \{0, 1, 2, 3\}$. In order to model the restrictions of this problem, a constraint is added for each pair of adjacent countries. If country x_i is adjacent to country x_j, then $x_i \neq x_j$ is required. The search algorithm is now responsible to select a variable and test a value of its domain. Assuming a simple "first variable, first value" strategy, the variable x_0 would be chosen and set to the value 0 as a test. This would be *propagated* to all variables which are directly linked to x_0 by a constraint, so that the value 0 gets removed from their domains. This removal may lead to other value removals in indirectly linked variables and is processed until a fix point is reached. If a contradiction is encountered or the domain of a variable becomes empty, *backtracking* is initiated, so that the next value of the variable x_0 is tested. Otherwise, the search algorithm continues with the next uninstantiated variable. This example also shows, that the propagation of the NotEqual constraint is *weak*, because it affects only two variables and invalidates only 4 out of the 16 possible value combinations between two variables.

The formulation of mapping problem as a Constraint Satisfaction Problem is not enough. Solutions for the variables in the CSP model need to be "interpreted" in the problem space. In the example above, the user needs to know, which country corresponds to x_i and which color corresponds to the value in its domain. Therefore, as a last step, an automatic transformation of the solution from the CSP model to the problem space is required. The entire process to automatically obtain a mapping solution is depicted in Figure 2.

B. Toolsuite ASSIST

As a proof of concept, the toolsuite *Architecture Synthesis for Safety-Critical Systems (ASSIST)* [19] was developed by the authors. It is open source and uses the constraint solver *Choco* [18] internally. ASSIST (see Figure 3) allows a systems engineer to automatically construct and optimize mappings based on textual specifications of the

- mappable elements,
- (safety) constraints for valid/invalid assignments and
- optimization goals.

The textual specifications in ASSIST conform to a domain-specific language which was jointly developed with the partners in the projects. This approach allows to hide the intricacies of a formal specification. Using a domain-specific language is expedient to enable systems engineers without a formal education in computer science to specify a mapping problem.

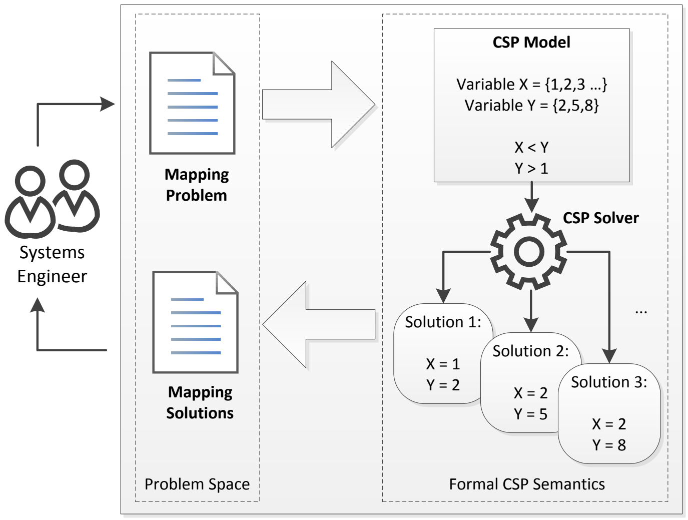
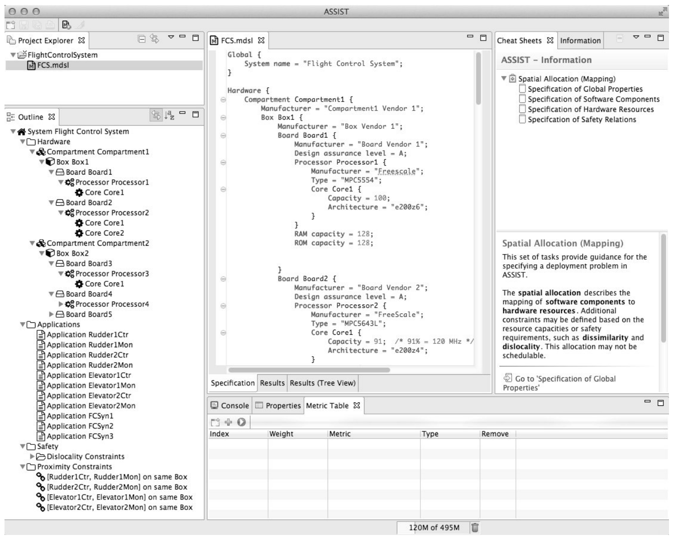


Fig. 2. Process to automatically construct a solution for a mapping problem



```
FlightControlSystem
  FCS.mdsl
```

```
System Flight Control System
  Hardware
    Compartment Compartment1
      Box Box1
        Board Board1
          Processor Processor1
            Core Core1
        Board Board2
          Processor Processor2
            Core Core1
            Core Core2
    Compartment Compartment2
      Box Box2
        Board Board3
          Processor Processor3
            Core Core1
        Board Board4
          Processor Processor4
        Board Board5
  Applications
    Application Rudder1Ctr
    Application Rudder1Mon
    Application Rudder2Ctr
    Application Rudder2Mon
    Application Elevator1Ctr
    Application Elevator1Mon
    Application Elevator2Ctr
    Application Elevator2Mon
    Application FCSyn1
    Application FCSyn2
    Application FCSyn3
  Safety
    Dislocality Constraints
  Proximity Constraints
    [Rudder1Ctr, Rudder1Mon] on same Box
    [Rudder2Ctr, Rudder2Mon] on same Box
    [Elevator1Ctr, Elevator1Mon] on same Box
    [Elevator2Ctr, Elevator2Mon] on same Box
```

```
Global {
    System name = "Flight Control System";
}

Hardware {
    Compartment Compartment1 {
        Manufacturer = "Compartment1 Vendor 1";
        Box Box1 {
            Manufacturer = "Box Vendor 1";
            Board Board1 {
                Manufacturer = "Board Vendor 1";
                Design assurance level = A;
                Processor Processor1 {
                    Manufacturer = "Freescale";
                    Type = "MPC5554";
                    Core Core1 {
                        Capacity = 100;
                        Architecture = "e200z6";
                    }
                }
                RAM capacity = 128;
                ROM capacity = 128;


            }
            Board Board2 {
                Manufacturer = "Board Vendor 2";
                Design assurance level = A;
                Processor Processor2 {
                    Manufacturer = "FreeScale";
                    Type = "MPC5643L";
                    Core Core1 {
                        Capacity = 91;  /* 91% = 120 MHz */
                        Architecture = "e200z4";
                    }
```

ASSIST – Information

Spatial Allocation (Mapping)
- ☐ Specification of Global Properties
- ☐ Specification of Software Components
- ☐ Specification of Hardware Resources
- ☐ Specifcation of Safety Relations

Spatial Allocation (Mapping)

This set of tasks provide guidance for the specifying a deployment problem in ASSIST.

The **spatial allocation** describes the mapping of **software components** to **hardware resources**. Additional constraints may be defined based on the resource capacities or safety requirements, such as **dissimilarity** and **dislocality**. This allocation may not be schedulable.

Go to 'Specification of Global Properties'

Index	Weight	Metric	Type	Remove

Fig. 3. Screenshot of ASSIST with a specification for a control system

IV. Experiences

By using ASSIST, the construction of valid solutions for large-scale mapping problems took less than 10 minutes on a regular desktop computer. In comparison to the multiple person months, which were previously required for a manual construction of a single solution, this represents a significant achievement. It not only improves the quality, but also the agility of the engineering process.

This achievement is a result of many experiments conducted by the authors in order to find an efficient CSP formulation and custom search strategies. The knowledge gained from these experiments will be discussed in this section and exemplified by the hypothetical spacecraft architecture. The experiences refer to the following aspects:

- Modeling a mapping problem and its constraints,
- Searching for a solution and
- Finding optimized solutions.

The knowledge gained in each of these aspects will be presented in the following subsections.

A. Modeling a Mapping Problem and its Constraints

1) Basic Model: Expressing the mapping problem of the present use case as a finite domain integer constraint satisfaction problem is generally straightforward. Each sensor is modeled with an integer variable whose values represent the index number of the assigned port. It is ensured that each port in the entire system has a globally unique index number. Initially, these sensor variables contain all possible port indices as their domain. With an additional pre-processing step, the domains for each variable can be further reduced by removing the index numbers of ports with an incompatible i/o type.

2) Only one sensor per port: Additional constraints for all of these variables must be defined to prevent multiple cables being assigned to the same port. This can be achieved with a global ALLDIFFERENT constraint [20] (with arc consistency) over all 3000 variables. Unfortunately, for large-scale mapping problems, using a *single* global ALLDIFFERENT constraint over so many variables is very computation-intensive – especially during backtracking. Instead it is necessary to further *partition* the sensor variables into multiple independent subsets. Sensors can be partitioned based on their unique i/o type, so that the application of an ALLDIFFERENT constraint for each of these disjoint subsets is sufficient. This approach drastically reduced the search time for a single solution.

3) Internally connected ports: Modeling the semantics of internally connected ports was initially tried by explicitly specifying valid or invalid mappings with a TABLE constraint [20]. However, this proved to be very inefficient for larger amounts of connected ports. The best alternative was to create $(n-1)$ "pseudo sensors" for each group of n internally connected ports. These "pseudo sensors" *blocked* the remaining ports after a "real" sensor has been mapped to any port of this group. For example, a group of four internally connected ports, requires three "pseudo sensors" to block the remaining unusable ports. The variables for each pseudo sensor were treated just like the other sensor variables. Only their domain was restricted to the indices of the ports in this port group.

4) Colocality of sensors: Up to this point, colocalities for sensors cannot be expressed, because hubs are not reflected in the model yet. Therefore, the authors decided to add a second variable for each sensor which contains the index of hub to which a sensor is assigned to. The "hub variables" and "port variables" of each sensor need to be linked to each other, so that the elimination of values in the domain of one variable also lead to the elimination of values in the domain of the other variable. Initially an ELEMENT [20] constraint was applied to express this relation. However, experiments showed that for large-scale mapping problems, modeling this relation by explicitly enumerating all allowed value pairs with a TABLE constraint and using the GAC3RM+ filtering algorithm [21], [22] performed better. Colocalities for sensors can now be expressed by requiring their "hub variables" to be equal. In contrast to other constraint solvers, such as *firstCS* [23], there is no ALLEQUAL constraint for more than two variables in *Choco*. There is only an ARITHM constraint which can be used to enforce equality between *two* variables. However, the semantics of an ALLEQUAL constraint can be emulated with a pairwise application of the ARITHM constraint. Propagation was found to be a little faster, if these pairs were posted like a star, instead of a list.

5) Dislocality of sensors: Dislocalities for a single group of sensors can be expressed by using a single ALLDIFFERENT constraint for the "hub variables" of all group members. Unfortunately, there is no similar constraint available to express a dislocality requirement for a *group* of sensor groups. It was necessary to develop a customized solution to prevent multiple groups of variables to share the same value, because sensors from different groups in a dislocality specification must not share the same hub. Initially, a custom constraint was implemented to simply check, if a value is contained in more than one group. However, the propagation of this approach was weak, because this check was only executed when all variables were instantiated, i.e., set to a single value.

In order to allow for a stronger propagation, the authors then tried to use an ALLDIFFERENT constraint for all variable tuples in the cross product of all sensor groups. Unfortunately, the sheer amount of sensor groups in dislocality specifications and the amount of sensors in each group required so many ALLDIFFERENT constraints, that a solution could not be found within the given timeframe.

With the goal to further reduce the amount of necessary ALLDIFFERENT constraints, the authors tried to use a *conflict graph*. In order to build this graph, a node was created for each "hub variable". An edge was added between two nodes, if it was determined, by analyzing the dislocality specifications, that the corresponding sensors must not share a hub. Once the entire graph was built, all *cliques* in the graph were located and an ALLDIFFERENT constraint was posted for each *clique*. This approach was implemented by using the *JGraphT* library [24], but unfortunately it did not turn out to be fruitful. In addition to the preprocessing effort of several minutes to create the conflict graph and search for all cliques, the reduced amount of ALLDIFFERENT constraints did not lead to a significant reduction of the search time.

Another constraint modeling approach was tried by the authors, which is inspired by the idea of creating a *set* for each sensor group and then requiring the intersection between all sets to be empty. Based on the implementation of a SETINTVALUESUNION constraint in Choco, which maintains a link between a *set variable* and the union of values of a list of integer variables, a custom INTVALUESUNION constraint was implemented. It ensures that a link between an *integer variable* and the union of values of a list of integer variables is maintained. With this constraint available, a variable for each group of sensors in a dislocality specification was created and linked to the union of the values of the "hub variables" in this group. This variable contained the hubs indices which are used by the sensors in this group. Expressing a dislocality specification between multiple groups now only required *one* ALLDIFFERENT constraint for all group variables, which are mentioned in a dislocality specification. Of course, these group variables have to be excluded from the search process, so that they are not instantiated to a single value, which would effectively require all sensors to be mapped to a single hub.

6) Breaking Symmetries: Solutions for large-scale mapping problems can be obtained fairly quickly as a result of the modeling approach described so far. Unfortunately, many of these solutions were largely similar. Analyzing a set of 5000 solutions showed, that more than 87% of all sensors were mapped to the same port in all solutions and that the remaining differences between these solutions resulted from permutations between mappings to similar ports of the same type within the same hub, so that all solutions resulted in the same cable length. The former aspect was addressed with specialized search algorithms (cf. section IV-B), but the latter aspect required an approach to break the following symmetry: two mapping solutions are assumed to be "equal", if the only differences are permutations of sensors within the same hub.

In research publications, it is often recommended to break symmetries by enforcing an artificial *order* of values, so that permutations are prevented. This was tried by adding constraints, so that on each hub, the sensors were required to be mapped to ports of the same type in an increasing order with respect to the index of the sensor. For example, for a hub with three similar ports (p_1, p_2, p_3) and three sensors (s_1, s_2,s_3) the only allowed assignment is ($s_1 \rightarrow p_1$, $s_2 \rightarrow p_2$, $s_3 \rightarrow p_3$).

Unfortunately, adding these constraints for all hubs resulted in many time-consuming backtracks during the search process before a solution, which satisfies this "increasing order" requirement, could be found. This approach may generally be well suited for small-scale mapping problems, but it proved to be ineffective for large-scale mapping problems.

In order to "break" this symmetry, the authors devised another solution. These permutations could instead be prevented by posting a constraint after a solution was found. This constraint declares a recent solution on the "hub variables" of each sensor as *invalid*, so that the same sensor to hub assignment would not be found again.

B. Searching

In large-scale mapping problems, the amount of possible solutions is extremely large, so that the entire solution space cannot be search through within reasonable time limits. Therefore, search strategies and heuristics need to be devised in order to reduce the search time. In Constraint Programming, search strategies consist of a *variable selector* and a *value selector*. The former determines, which of the uninstantiated variables will be tried to instantiate next (e.g. which sensor to choose for the next mapping step). The latter chooses a value from the domain of the selected variable for constraint propagation and satisfiability checking. Therefore the development of a custom search strategy requires a custom variable selector and a custom value selector.

1) White-box and black-box strategies: Search strategies based on heuristics can be generally classified as being either *white-box* or *black-box*. White-box strategies try to exploit domain knowledge for selecting variables and values, while black-box strategies exploit only "generic" properties of the variables and their associated constraints. At first, the authors tried to develop efficient white-box strategies by extensively interviewing the systems engineers who previously conducted the manual mapping. Unfortunately, this approach was not conducive to develop an efficient variable selector, because the engineers did not select sensors on individual level, but instead selected all sensors of an entire subsystem and mapped these groups to a hub in the vicinity. This coarse-grained approach failed to tap into the optimization potential of spreading subsystems over multiple hubs by mapping sensors individually.

Several state-of-the-art black-box variable selection strategies were tried instead. Variables were selected based on *minimum domain size (*MINDOM*)*, *minimum domain size over weighted degree (*DOMWD*)*, *variable activity* and *variable impact*. MINDOM chooses variables with the least flexibility, i.e., the smallest domain, first. In the example, this strategy chooses the sensors with the least amount of available ports first. DOMWD on the other hand, chooses variables with the least flexibility and which are affected by the highest amount of constraints first. These two approaches yielded the best results in the example. The other black-box strategies based on the concepts of *variable activity* during propagation [25] and the *impact* of a variable for the reduction of the search space [26] have been tried with their default parameters in Choco, but they did not perform as well as DOMWD or MINDOM.

Unfortunately, due to the size of the problem and the partial randomness in the successful black-box strategies, it was not possible to completely trace and comprehend the reason, *why* DOMWD outperformed the white-box variable selection strategies. Moreover, no valuable recommendations could be extracted for the systems engineers regarding the order in which sensors should be mapped in order to achieve a mapping solution.

In contrast to the experiences gained from implementing custom *variable* selectors, the exploitation of domain knowledge was key to create effective white-box *value* selectors. Initially, the authors tried common black-box value selectors, such as MINVALUEFIRST, MAXVALUEFIRST and RANDOMVALUE. The latter resulted in mapping solutions within minutes, but the resulting cable weight of the entire system left a lot of room for improvement, because sensors were attached to randomly chosen ports. The other two black-box value selectors did not result in a solution within a 60 minute time frame, because sensors were tried to be concentrated on as few hubs as possible, which resulted in a lot of backtracking. Based on these observations, a white-box value selector was implemented. This CLOSESTPORTFIRST selector prefers values representing ports, which have the smallest geometrical distance to the currently selected sensor. It was very effective in the example and lead to optimized mapping solutions as it distributed all sensors over all hubs in the system, so that fewer backtracks were required. Furthermore, it incorporated the information about the distance between hub and sensor in order to achieve a reduced cable length.

2) Feedback for infeasible solutions: With the goal to support a design space exploration of the system, the architecture of the hubs was modified by the systems engineers with regard to the amount of hubs being available and the type of ports being offered. For a given architecture, the successful construction of a mapping was then taken as a basic "feasibility test". The architectures, for which the construction of a mapping was tried, were close to the border between being feasible and infeasible. Subtle changes lead to significant and unforeseen consequences and rendered the mapping problem unsolvable.

Therefore, if a solution could not be found within a given time limit, the system engineers needed to get feedback regarding bottlenecks in the architecture. Is another hub needed to satisfy all dislocality requirements? Or is it sufficient to add another port of a specific type to a hub?

Unfortunately, it is difficult to address both needs at the same time: find a solution quickly if the problem is feasible *and* try to map as many sensors as possible to the hubs in order to determine bottlenecks in the architecture. This is a result of the *fail fast* nature of most search strategies.

Fail fast search strategies try to crack the "hardest nut" first. The hardest nut often refers to finding an assignment for the variables with the smallest degree of freedom and the most restrictions. Given an unsolvable mapping problem, these strategies will fail quickly after assigning only a few sensors. In ASSIST, these strategies returned to the user with a message like "*2% of the sensors were successfully assigned before a contradiction was encountered*". Unfortunately, this message is misleading for systems engineers. They *assumed*, that only 2% of the sensors *could* be mapped due to a limited amount of hubs in the system. The engineers then tested a RANDOM strategy – without fail fast nature – and complained, that it appeared to be more successful in comparison to state-of-the-art black-box strategies. Helpful feedback would instead have been the information, that, for instance, "*94% of all sensors were assigned successfully, but there are conflicts with the remaining 6% of the sensors*". Unfortunately, the application of alternative *fail late* strategies failed to produce mapping solutions for cases in which *fail fast* strategies were successful. The engineers thus had to check with a *fail fast* strategy, if an architecture is feasible and apply an *fail late* strategy, if it was not feasible, in order to determine bottlenecks.

Experiments showed, that a RANDOM strategy as a *fail late* strategy was of limited use for the engineers. Analyzing a partial solution of *randomly* chosen sensors did not help significantly in identifying bottlenecks or using it for an incremental mapping approach. Therefore, the authors developed a hybrid search strategy by combining the effectiveness of a black-box strategy with the valuable feedback gained from partial solutions of white-box strategies. It requires the systems engineers to manually assign search priorities, e.g. 1 (highest) to 5 (lowest), to all sensors. The variable selector initially groups all of the sensor variables based on their search priority and then selects the variables within each group based on the DOMWD strategy. If a contradiction is reached during search, this approach ensures, that the partial solution contains the mapping of the sensors with the highest priorities. These partial results could then be used as part of an *incremental* mapping process. However, this hybrid approach did not reach the efficiency of the plain DOMWD strategy and its performance significantly depended on the manual assignment of search priorities and the amount of priority classes.

C. Optimized Solutions

Solutions to the sensor mapping problem should be optimized with respect to the minimization of the total cable weight. The total cable weight corresponds to the sum of the cable weight for each sensor. Each of these individual cable weights is calculated based on a constant *weight factor* and the geometrical distance between sensor and hub. In reality, the weight factor is not constant. Instead, it depends on the distance, because electromagnetic shielding requirements are increasing with longer sensor cables, so that the weight factor increases as well.

In order to get optimized solutions, the authors initially created a set of 10000 valid solutions and automatically evaluated each solution with respect to the total cable weight. This approach would allow the systems engineers to get a feel for the solution space and to manually select the best fitting solution – even if it is not the one with the lowest total cable weight.

However, in large-scale mapping problems, these 10000 solutions represented only a tiny fraction of the space of possible solutions. Furthermore, they were found to be very similar to each other. Because of the backtracking approach in the search strategies in order to *systematically* go through all possible value combinations, they differed only in very few assignments. In order to increase the variety of solutions, the authors applied a technique called "restarts", so that the search process is restarted from the root of the search tree, if a solution is found or if a configurable amount of *fails* has been reached by traversing the search space. This approach required randomness in the search strategies to avoid finding similar solutions twice, but it helped to get significantly different solutions. Therefore, the likelihood of finding "better" solutions increased as well. The experiments showed, that triggering a restart after reaching 400 *fails* yielded the best results.

The minimum total cable weight within these 10000 results still left room for optimization. In addition to simply increasing the likelihood for finding a better solution, the authors used the mechanisms offered by Choco to *enforce* a minimization during search. For this purpose, a "weight variable" was added for each sensor, which captures its cable weight as a result of its assignment. A single "total weight variable" is added as well, which represents the sum of all "weight variables".

With these additional variables being available, a constraint for the "total weight variable" can be posted after a solution is found. This constraint ensured, that solutions yielding a higher total cable weight are invalidated. Enabling the enforced optimization lead to significantly better results.

V. Conclusions and Outlook

Large-scale mapping problems in real-world systems with thousands of variables and tens of thousands of constraints *can* be automatically solved within minutes on regular desktop computers. The automation, which is described in this contribution, constitutes a significant productivity gain and increases the agility, efficiency and quality of the engineering process. This approach requires an efficient formalization of the mapping problem and customized search strategies – both of which are well supported with Constraint Programing.

For future versions of ASSIST, the authors plan to further increase the benefits of an automated mapping process by improving the feedback that can be gathered from analyzing partial solutions or based on solver statistics of intractable mapping problems. Safety constraints and resource bottlenecks, which prevent finding a solution, should be easier to locate for the systems engineer. Furthermore, the support for an incremental mapping process will be improved as well. It should allow the systems engineer to fix the successful mapping of certain sensors and to retry the mapping of the remaining sensors with a slightly modified hub architecture.

Acknowledgments

The authors acknowledge the financial support for this work by the Federal Ministry of Education and Research of Germany (BMBF) in the project "ARAMiS II" (grant identifier 01IS16025D).

References

[1] T. Gaska, B. Werner, and D. Flagg, "Applying virtualization to avionics systems - The integration challenges," in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, Oct. 2010, pp. 5.E.1–1 – 5.E.1–19.

[2] E. Lee, "Cyber Physical Systems: Design Challenges," in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, May 2008, pp. 363–369.

[3] P. Derler, E. Lee, and A. Vincentelli, "Modeling Cyber-Physical Systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, Jan. 2012. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/843.html

[4] A. Hall and R. Chapman, "Correctness by Construction: Developing a Commercial Secure System," *IEEE Software*, vol. 19, pp. 18–25, Jan. 2002. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/52.976937

[5] R. Chapman, "Correctness by construction: a manifesto for high integrity software," in *Proceedings of the 10th Australian workshop on Safety critical systems and software - Volume 55*, ser. SCS '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 43–46. [Online]. Available: http://dl.acm.org/citation.cfm?id=1151816.1151820

[6] S. M. Leinwand, "Logical correctness by construction," in *Proceedings of the 19th Design Automation Conference*, ser. DAC '82. Piscataway, NJ, USA: IEEE Press, 1982, pp. 825–831. [Online]. Available: http://dl.acm.org/citation.cfm?id=800263.809296

[7] M. Bordin and T. Vardanega, "Correctness by construction for high-integrity real-time systems: a metamodel-driven approach," in *Proceedings of the 12th international conference on Reliable software technologies*, ser. Ada-Europe'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 114–127. [Online]. Available: http://dl.acm.org/citation.cfm?id=1769168.1769177

[8] R. Hilbrich, "How to Safely Integrate Multiple Applications on Embedded Many-Core Systems by Applying the "Correctness by Construction" Principle," *Advances in Software Engineering*, vol. 2012, no. 354274, p. 14, 2012. [Online]. Available: http://www.hindawi.com/journals/ase/2012/354274/

[9] R. Hilbrich and L. Dieudonne, "Deploying Safety-Critical Applications on Complex Avionics Hardware Architectures," *Journal of Software Engineering and Applications*, vol. 06, no. 05, pp. 229–235, 2013. [Online]. Available: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=31297

[10] W. Damm, A. Metzner, F. Eisenbrand, G. Shmonin, R. Wilhelm, and S. Winkel, "Mapping Task-Graphs on Distributed ECU Networks: Efficient Algorithms for Feasibility and Optimality," in *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, 2006, pp. 87–90.

[11] S. Kugele, W. Haberl, M. Tautschnig, and M. Wechs, "Optimizing Automatic Deployment Using Non-functional Requirement Annotations," in *Leveraging Applications of Formal Methods, Verification and Validation*, ser. Communications in Computer and Information Science, T. Margaria and B. Steffen, Eds. Springer Berlin Heidelberg, 2009, vol. 17, pp. 400–414. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88479-8_28

[12] S. Voss and B. Schatz, "Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications," in *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*, 2013, pp. 100–109.

[13] J. White, B. Dougherty, C. Thompson, and D. C. Schmidt, "ScatterD: Spatial deployment optimization with hybrid heuristic/evolutionary algorithms," *TAAS*, vol. 6, no. 3, p. 18, 2011.

[14] J.-L. Lauriere, "A language and a program for stating and solving combinatorial problems," *Artificial Intelligence*, vol. 10, no. 1, pp. 29–127, 1978, cited By 124. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0002931606&doi=10.1016%2f0004-3702%2878%2990029-2&partnerID=40&md5=107f5b639c293243c04e786c3ca3917e

[15] K. R. Apt, *Principles of constraint programming*. Cambridge University Press, 2003.

[16] R. Dechter, *Constraint Processing*. Elsevier Science & Technology, 2003. [Online]. Available: http://www.ebook.de/de/product/3261979/rina_dechter_constraint_processing.html

[17] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. ELSEVIER SCIENCE & TECHNOLOGY, 2006. [Online]. Available: http://www.ebook.de/de/product/5834373/handbook_of_constraint_programming.html

[18] C. Prud'homme, J.-G. Fages, and X. Lorca, *Choco Documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. [Online]. Available: http://www.choco-solver.org

[19] R. Hilbrich, "Architecture Synthesis for Safety Critical Systems - ASSIST," online, 2014. [Online]. Available: http://assist.hilbri.ch

[20] N. Beldiceanu, M. Carlsson, and J.-X. Rampo, "Global Constraint Catalog," online, SICS, Technical Report T2012:03, Feb. 2014, iSSN: 1100-3154. [Online]. Available: http://www.emn.fr/z-info/sdemasse/gccat/

[21] C. Lecoutre and F. Hemery, "A study of residual supports in arc consistency," in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007, pp. 125–130. [Online]. Available: http://ijcai.org/Proceedings/07/Papers/018.pdf

[22] A. Paparrizou, *Efficient Algorithms for Strong Local Consistencies and Adaptive Techniques in Constraint Satisfaction Problems*. LULU Press, 2015.

[23] A. Wolf, "Object-Oriented Constraint Programming in Java Using the Library firstcs," in *WLP*, 2006, pp. 21–32.

[24] "Welcome to JGraphT - a free java graph library." [Online]. Available: http://jgrapht.sourceforge.net/

[25] L. Michel and P. Van Hentenryck, "Activity-based search for black-box constraint programming solvers," in *Proceedings of the 9th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, ser. CPAIOR'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 228–243. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29828-8_15

[26] P. Refalo, *Impact-Based Search Strategies for Constraint Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 557–571. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30201-8_41