3D State Decomposition Modeling Method for MBSE in Space Engineering

Hao Zhang*

German Aerospace Center (DLR), Simulation and Software Technology, Braunschweig, 38108,Germany
Center for Space Science and Applied Research, Chinese Academy of Sciences, Beijing, 100190, China
Graduate School of Chinese Academy of Sciences, Beijing, 100190, China
Email: hao.zhang@dlr.de

Andreas Gerndt†

German Aerospace Center (DLR), Simulation and Software Technology, Braunschweig, 38108,Germany
Email: andreas.gerndt@dlr.de

Bo Liu‡

Center for Space Science and Applied Research, Chinese Academy of Sciences, Beijing, 100190, China
Email: boliu@nssc.ac.cn

The increasing number of space missions and the market environment make it mandatory that the design process and design models have to be reused in future space missions. As a solution to such a reuse problem, Model-Based Systems Engineering (MBSE) is highly recommended. Since constructing a model is the first step to reach MBSE, a 3D state decomposition modeling method is proposed in this paper. Starting from Technical Requirements (TRs), the method maps them to state variables and defines states as well as relations along three red lines of space engineering: internal subsystem, inter-subsystems, and space mission lifecycle. Eventually, it yields decomposition of a space system into a hierarchical three-dimensional model of states. Relying on the Eclipse Modeling Framework (EMF), the model can be described and stored in a standardized format, which allows the reuse of once defined models in subsequent space missions. This study leads to a novel MBSE solution to implement specification, analysis, design, and operation of space mission systems in a unified framework.

I. Introduction

There are more and more space missions. This requires development lifecycle to be decreased and development cost to be reduced, but to keep the high quality of the products at the same time. To resolve this conflict, different strategies are possible:

- Optimization of design models: If design models are optimized as early as possible, then the iterative times throughout the whole design process may be decreased. Therefore, the development cost and lifecycle can be reduced whereas the product quality can be increased even more.
- Reuse of qualified development products: If we can reuse qualified development products such as documents, simulation models, design models, and development processes etc., then we do not need to start everything from scratch again and again. Therefore, the time and cost required to develop a new product will be saved. On the other hand, the product quality can still be assured.

*Simulation and Software Technology, Lilienthalplatz 7, 38108 Braunschweig, Germany.
†Head of Department, Simulation and Software Technology, Lilienthalplatz 7, 38108 Braunschweig, Germany.
‡Deputy Director, NO.1 Nanertiao, Zhongguancun, Haidian district, Beijing, 100190, China.

- Reduction of overheads: Since a space mission is costly and has only one launch chance, many quality assuring measures are taken to prove that the whole development process is going well. For example, the whole development process is divided into several phases with reviews in-between.[7] Engineers not only have to achieve the goals of each phase, but also have to write a large number of documents for these reviews, which causes high overheads for them. If we can reduce these overheads, the development time and cost can also be saved.

Some solutions have been proposed from different point of views. Multidisciplinary Optimization (MDO), for instance, is a research example to optimize a multidisciplinary design solution. It is defined by Sobieszczanski-Sobieski as "a methodology for the design of systems where interaction between several disciplines must be considered, and where the designer is free to significantly affect the system performance in more than one discipline".[2] In order to perform optimization algorithms, mathematical design models have to be integrated together because they are tightly coupled. However, in order to decrease the computation burden, the models used for multidisciplinary optimization are less complex and less accurate than the models used for single discipline optimization.[2] Furthermore, MDO lacks "a full life cycle business case" because it starts at a stage where the design is already very detailed.[5] If we want to apply MDO to the whole lifecycle of a space mission, we need to perform further research.

High Level Architecture (HLA) is a research example to facilitate the reuse of simulation systems and simulation-related assets.[1] In the HLA, a simulation system is called a federation. Multi-simulation applications, which are called federates, are connected together to construct a distributed execution environment. There are two important simulation assets in a HLA-based simulation system: federation object model (FOM) and simulation object model (SOM). FOM is to provide a specification of data exchange among federates whereas a SOM is an interface specification for each federate in the federation.[3] However, HLA can not make a FOM or a SOM "plug and play".[4] The reason is that only data needed to be exchanged among federates are defined in a SOM or a FOM. In fact, such a data definition is highly related with the requirements of a current simulation. In order to reuse a SOM or FOM, the simulation requirements of a former federate or federation have to be throughly investigated. Therefore, it is difficult to reuse a HLA-based simulation system or simulation-related assets.

All solutions proposed up to now have some shortcomings. Some of these are addressed by our approach presented in this paper. We focus on the following aspects:

- Traceability of requirements: Requirements are the baseline to develop a space system. If we want to reuse development products generated throughout a development process, we have to look into the requirements behind them. For example, a SOM or a FOM in the HLA cannot be traced back to requirements. This means, we can not make a decision whether it can be reused or not only based on data definitions. Therefore, in order to successfully implement the reuse of development products, our approach is to provide the ability to trace back to requirements.

- Automatic documentation generation: The high overheads for engineers mainly come from writing and maintaining many documents. Therefore, our approach pursues the possibility to generate documents automatically.

- Optimization of design models throughout a complete space mission lifecycle: If we can get optimized design models on each design phase and level, product quality will be assured to the maximum degree. As mentioned above, MDO can not perform this task yet. Therefore, our approach will also work on it.

To achieve these goals, we are working on a solution following Model-Based Systems Engineering (MBSE), which incorporates the three-dimensional scope of systems engineering: the full lifecycle, the full depth, and the full technical scope.[9] Since constructing a model is the first step to reach MBSE, our solution starts from proposing a modeling method, which will be presented in this paper. The remainder of this paper is organized as follows. Section II describes the characteristics of a space system whereas its development process is illustrated in Section III. Section IV focuses on the requirements of a model for MBSE and briefly lists current research work in MBSE in the space domain. Section V describes the proposed modeling method, which includes modeling steps and modeling software environment. In Section VI, we outline research directions for the future work.

II. Characteristics of a Space System

A space system is decomposed into subsystems, which interrelate with each other. For example, in Fig. 1, a sounding rocket based exploration system comprises a sounding rocket subsystem, a payload subsystem, an on-board data management subsystem, a data receiver subsystem, a raw data processing subsystem, a science data processing subsystem, a control application etc.

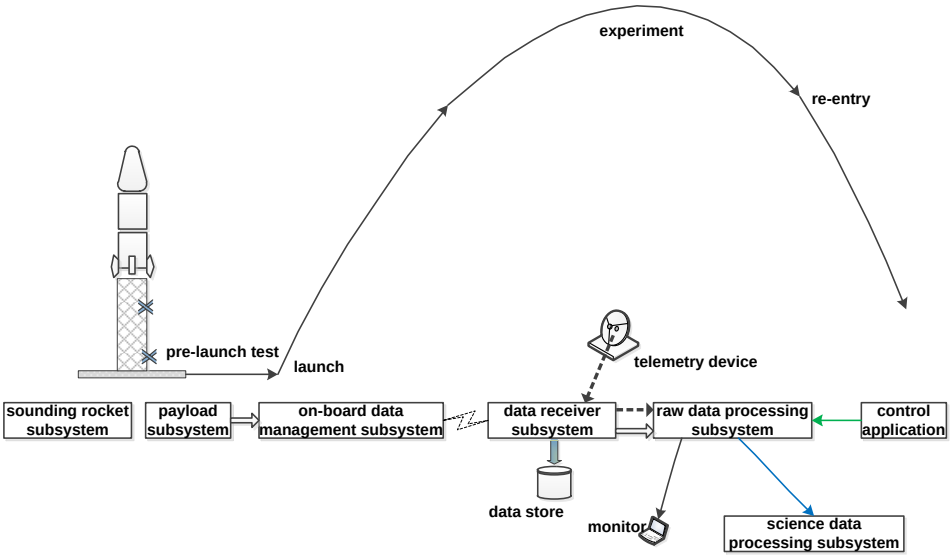

Figure 1. This is a data flow diagram among subsystems of the sounding rocket based exploration system. The lifecycle of the space mission is divided into the pre-launch test phase and the after-launch phase, which comprises the launch sub-phase, the experiment sub-phase, and the re-entry sub-phase. Dependency among the subsystems may be different in the different phases.

In this paper, we examine the raw data processing subsystem as an example. It is assigned to fulfill the following functions:

- Reading received down-link data from the data receiver subsystem.
- Unpacking, processing, and displaying payload data.
- Generating raw data packages for the science data processing subsystem.
- Displaying the trajectory and fly events after the sounding rocket is launched.
- Monitoring the telemetry device states.

In order to fulfill these functions, the raw data processing subsystem needs to know about the following data format specifications used in other subsystems:

- data receiver subsystem: In order to read received down-link data from the data receiver subsystem, the communication protocol of such data has to be specified. For the monitoring of telemetry device states, the communication protocol of state data has to be specified by the data receiver subsystem.
- payload subsystem: In order to unpack, process, and display payload data, data package format has to be defined by the payload subsystem.
- science data processing subsystem: In order to transfer raw data packages to the science data processing subsystem, the requirements of raw science data product format have to be specified.
- control application: Since the sounding rocket subsystem has no capabilities to transmit fly events, a control application offers additional information about the mission which can augment the visualization of payload data, e.g. trajectory information. The format of these data has to be defined.

III. Development Process of a Space System

A. Lifecycle Definition in the Space Domain

As shown in Fig. 2, the ***lifecycle of a space mission*** is typically divided into seven phases: Phase 0, Phase A, Phase B, Phase C, Phase D, Phase E, and Phase F.[6]

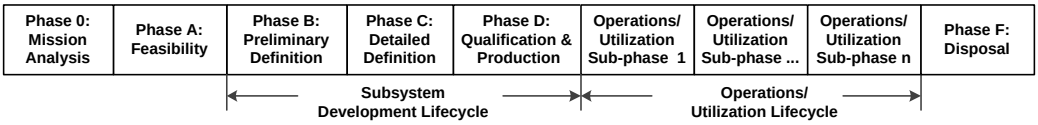

Figure 2. This figure shows a typical lifecycle of a space mission. Phase B, together with Phase C and Phase D are defined as the subsystem development lifecycle. Phase E is defined as a system-level operations/utilization lifecycle.

At the end of Phase A, technical requirements (TRs), which "express the customer's need, mission statement, associated environmental constraint and programmatic element", are established and assigned to subsystems. These are the "business agreement for the next phase".[7]

For each subsystem, its ***development lifecycle*** starts from Phase B and ends at the end of Phase D. During this process, the decomposition from TRs to components until implementation is carried out, considering the relations between the subsystem and other subsystems. For example, as can be seen in Fig. 3, a data reader component, a data processing component, a data display component etc. can be decomposed from the raw data processing subsystem's TRs. For the data reader component, a data reader unit, a data store unit, and a data forwarding unit can be further decomposed. Such decomposition work continues until each component is implemented as programs.[8] Meanwhile, the external relations with the other subsystems have to be clearly specified.

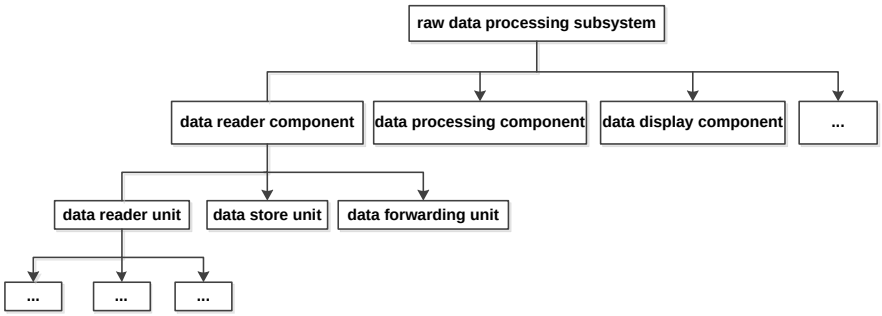

Figure 3. This figure shows decomposition of the raw data processing subsystem into its essential parts. The decomposition is refined with the development process advances. Implementation of the subsystem is achieved in the end.

At the end of Phase D, the subsystem product is delivered[8] and commences the ***operations/utilization lifecycle***, i.e. Phase E. "Phase E comprises all activities to be performed in order to launch, commission, utilize, and maintain the orbital elements of the space segment and utilize and maintain the associated ground segment". Taking a software product as an example, "the operation process can start after completion of the acceptance" and its "phasing and management are determined by the overall system requirements".[8] Therefore, the "major task for Phase E varies widely as a function of the type of project concerned".[7]

An operations/utilization lifecyle is divided into several ***operations/utilization sub-phases***. For example, for a satellite space mission with on-orbit operations, Phase E may contain a system-level ground integration test sub-phase and different on-orbit operation modes. However, the sounding rocket based exploration mission mentioned above has a launch sub-phase, an experiment sub-phase, and a re-entry sub-phase after the sounding rocket is launched. Therefore, its Phase E is divided into system-level ground integration test sub-phases and the three after-launch sub-phases. According to its system-level requirements, three ground integration test sub-phases are defined: the desktop-wired test sub-phase, the desktop-wireless test sub-phase, and the pre-launch test sub-phase. Therefore, for the sounding rocket based exploration system example, six operations/utilization sub-phases are defined in Phase E: the three pre-launch test sub-phases

and the three after-launch sub-phases.

But, for the raw data processing example, since its state in the three after-launch sub-phases is the same, it has only four ***usage sub-phases*** throughout the space mission lifecycle: the three pre-launch test sub-phases and the after-launch sub-phase. ***Usage sub-phases*** represent the ways for a subsystem or its component to be used. Specifically, for the raw data processing subsystem, aside from the common functions, the uniqueness for each usage sub-phase is:

- Desktop-wired test sub-phase: In this sub-phase, the hardware interface between the data receiver subsystem and the raw data processing subsystem is a Universal Serial Bus (USB) device. The raw data processing subsystem should have the ability to access it and read received down-link data.
- Desktop-wireless test sub-phase: In this sub-phase, the hardware interface between the data receiver subsystem and the raw data processing subsystem is a wireless network card. The raw data processing subsystem should switch the interface from a USB device to a wireless network card.
- Pre-launch test sub-phase: In this sub-phase, the data receiver subsystem involves the telemetry device. The raw data processing subsystem should have the ability to read received down-link data and the telemetry device states via serial communication protocol.
- After-launch sub-phase: Aside from fulfilling all of the functions in the pre-launch test sub-phase, the trajectory and fly events of the sounding rocket should be visualized. And the raw data packages should be transmitted to the science data processing subsystem.

Before the raw data processing subsystem is delivered, the four usage states are performed on subsystem level to complete the subsystem-level test. After it enters Phase E, the four usage states are performed again, but together with other subsystems on system-level operations. In Phase F, there is no special state requirements. Therefore, the four usage sub-phases cover all of its usage state requirements throughout the space mission lifecycle. Engineers have to take these state requirements into consideration when they design and implement the subsystem. After the mission was performed, the lifecycle enters Phase F, which is the implementation of the disposal plan.[6]

B. Traditional Document-based Development Lifecyle

To develop a space system, we have to focus on the development lifecycle of a subsystem because it is decomposed into different subsystems. As mentioned above, the subsystem development lifecycle starts from the beginning of Phase B and ends at the end of Phase D. To ensure that specified objectives are achieved, "each of the phases includes milestones in the form of project reviews, the outcome of which determines readiness of the project to move forward to the next phase".[6] As can be seen in Fig. 4, the reviews, e.g. System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), Qualification Review (QR), and Acceptance Review (AR)[6] are set down throughout the subsystem development lifecycle.

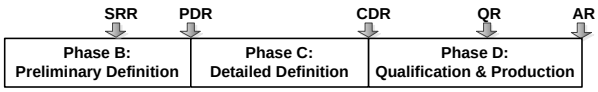

Figure 4. This figure shows reviews in a subsystem development lifecycle. Proof of the current development process has to be provided for these reviews. In traditional document-based engineering process, the proof is in documents.

In traditional document-based development lifecycle, a large number of documents must be manually prepared and maintained for these reviews. The documentation contains system specification, interface requirement document, design document, user manual, interface control document, validation plan, operational plan etc.,[8] which split information of a development lifecycle into pieces. They are generated by different engineers, for example, the validation plan is written by quality assurance engineers, who are independent of the developers. Therefore, controlling the versions of the documentation becomes very troublesome. Thus, such a document-based approach bears the risk of an inconsistency among the different documents. The

consequence is that it is difficult to integrate the appropriate documents together and reconstruct a development lifecycle. It is even more difficult to implement the reuse of correct intermediate products or the development lifecycle.

IV. Model-Based Systems Engineering (MBSE)

A. Requirements of a Model for MBSE

As an approach to resolve the problems mentioned above, MBSE integrates information of a development lifecycle into a model. Such a model should address the following aspects:

- It should have the ability to trace back to TRs. TRs are the baseline to develop a subsystem and they may be changeable throughout the whole development process. With the traceability back to TRs, it is possible to detect requirement changes and corresponding measures could be taken. Another benefit is that we can determine quickly that a model can be reused on condition that the TRs are the same as those of the current subsystem.

- The model should represent different usage sub-phases throughout the lifecycle of the space mission. To achieve various usage states is the ultimate purpose to develop a subsystem. These usage sub-phase states should be explicitly defined from the very beginning of the development process. Some of them have been defined in TRs. Some of them need to be identified according to the possible usage states throughout the development lifecycle, operations/utilization lifecycle, and Phase F. Such identification needs engineers' experience.

- The relations between the subsystem and other subsystems should be specified. As described in Section II, the subsystems in a space system interrelate with each other. When we design and implement a subsystem, it is impossible to ignore its relations with other subsystems. Therefore, inter-subsystem relations have to be explicitly described in the model.

- The model can be refined throughout the subsystem development lifecycle. At the beginning of Phase B, only TRs are specified for a subystem. At the end of the development lifecycle, the subsystem is completely implemented. Therefore, the granularity of the model changes from a very high level to the bottom level. Thus, the internal states and relations inside the subsystem should be illustrated and maintained in the model.

B. Current Work in MBSE

The main concept of MBSE is to rely on a consistent model throughout the whole engineering process. The concept can be used in any domain such as automobile, energy, and space. The goal of MBSE in the space domain is to implement specification, analysis, design, and operations of a space mission in a unified framework. "Virtual Satellite",[10] being developed by the Simulation and Software Technology at DLR, Germany, is such a framework. There are also several other solutions. One approach is to develop a space engineering standard such as the ECSS standard.[11] It is based on the concept that "the key item is data". It defines a mechanism whereby engineering data can be reliably and efficiently exchanged.[11] An ideal implementation of this approach is expected as a space system data repository.[12] Another approach is state analysis.[13] In state analysis methodology, a process for capturing system and software requirements in the form of explicit models is defined. "A state represents a momentary condition of an evolving system and a model describes how the state evolves and is affected by others".[13] Mission Data System (MDS), "an embedded software architecture that leverages the state analysis methodology",[14] is currently under development at the Jet Propulsion Laboratory (JPL).

In order to construct a model for MBSE, we will present a modeling method in Section V. It is called 3D state decomposition modeling method, which also follows the state analysis methodology.

V. 3D State Decomposition Modeling Method

A. Hypothesis

In model-based space engineering process, the traditional development lifecycle is regarded as a model refinement process from Phase B to Phase D. As can be seen in Fig. 5, in accordance with the synchronized review points, the model is marked as a System Requirement Review (SRR) level model, a Preliminary Design Review (PDR) level model, a Critical Design Review (CDR) level model, a Qualification Review (QR) level model, or a Acceptance Review (AR) level model. Information distributed in different kinds of documents throughout traditional development lifecycle is expected to be integrated into the appropriate level model.

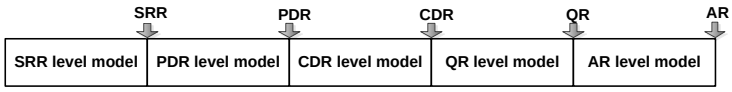

Figure 5. This figure shows reviews in model-based development lifecycle. Information of a development process is integrated into a model. The model is refined with the development process goes on, also with reviews in-between.

B. Modeling Process

For a subsystem in a space system, the modeling work starts from its Technical Requirements (TRs). TRs are established at the end of Phase A and "exclude requirements such as cost, methods of payment, quantity, time or place of delivery".[7] In the modeling method, three dimensions (3D): the lifecycle of a space mission, the separation into internal subsystems, and relations of inter-subsystems in-between are considered.

As shown in Fig. 6, based on TRs, firstly, all of the TRs are translated to variables describing certain subsystem states. When this is done, the relations between these subsystem state variables are identified along the three dimensions. After the decomposition was carried out on the subsystem level, further identification of state variables is performed for each state variable. Such decomposition work ends when current decomposition meets the requirements for the model. For example, the information that a SRR level model carries should meet the information requirements from the System Requirement Analysis phase, which is a phase in Phase B.[8] At least, it should be enough to prepare for a SRR review. The further decomposition of a state variable may also stop when it can be implemented as programs.

In order to illustrate the method, we still take the raw data processing subsystem of the sounding rocket based exploration system mentioned in Section II as an example.

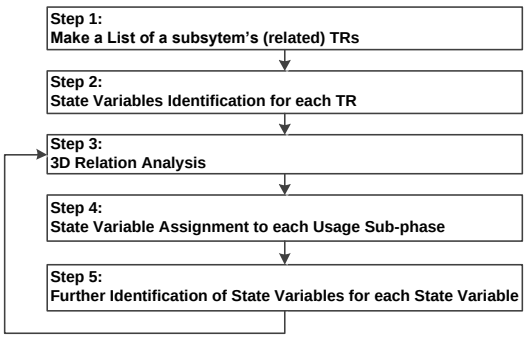

Figure 6. This figure shows the 3D state decomposition modeling process. The level of the decomposition depends on the current development phase.

1. *Step 1: Make a List of a subsytem's (related) TRs*

The first step is to list all TRs of the subsystem. TRs can be divided into System Requirements (SRs) and Related System Requirements (RSRs). SRs are requirements that the subsystem should fulfill, such

TR Type	ID	Requirement Text
SRs	SR1	The raw data processing subsystem should read received down-link data from the data receiver subsystem.
	SR2	The raw data processing subsystem should unpack, process, and display payload data.
	SR3	The raw data processing subsystem should generate raw data packages for the science data processing subsystem.
	SR4	The raw data processing subsystem should display the trajectory and fly events after the sounding rocket is launched.
	SR5	The raw data processing subsystem should monitor the telemetry device states.
RSRs	RSR6	The trajectory and fly event parameters will not be defined until one hour before the launch.
	RSR7	There is only one telemetry device in this mission.
	RSR8	Payload data package should meet CCSDS standard.
	RSR9	There are three payloads in the mission: payload A, B, and C.
	RSR10	The down-link data rate is 2M byte/s.

Table 1. This is a list of TRs of the raw data processing subsystem. TRs are divided into SRs and RSRs. SRs are the requirements that it has to fulfil. RSRs are the constraints that have to be considered when meeting these requirements.

as functional requirements, performance requirements, operational requirements etc. RSRs are constraints that have to be considered when implementing SRs. As required in the ECSS standard,[7] a unique identifier number is assigned to each of them. For the raw data processing subsystem example, its SRs and RSRs are listed in Table. 1. The SRs are the functions defined in Section II, and the RSRs are the constrains when performing them. For example, RSR6 is a prerequisite for the implementation of SR4. As descirbed in SR4, the trajectory and fly events will be visualized after the sounding rocket is launched. In order to fulfill this requirement, a user interface should be provided so that the data can be updated one hour before the launch, as described in RSR6.

2. Step 2: State Variables Identification for each TR

As mentioned in Step 1, the list of TRs consists of SRs and RSRs. At this step, they are translated to variables describing certain subsystem states. State variables of a SR or a RSR may contain Requirement State Variables (ReqSVs), or Inter-subsystem State Variables (InterSVs), or both. A ReqSVs indicates a state that can represent a SR or a RSR. An InterSV represents a relation between the subsystem and another subsystem. Again, a unique identifier number is given to each of them so that it is possible to be globally retrieved, which is crucial in subsequent model-based simulation. The work is carried out manually by the engineers, based on their experience. Taking "SR1: The raw data processing subsystem should read received down-link data from the data receiver subsystem" as a SR example, its ReqSV is identified and named as ReqSV1.

- ReqSV1: The raw data processing subsystem should have the ability to read received down-link data from different external hardware interfaces of the data receiver subsystem.

Taking "RSR6: the trajectory and fly event parameters will not be defined until one hour before the launch" as a RSR example, its state variables consist of one ReqSV and one InterSV. The ReqSV is named as ReqSV8. The InterSV is named as InterSV8-1, which represents an inter-subsystem relation between the raw data processing subsystem and the control application.

- ReqSV8: The raw data processing subsystem should have the ability to update the trajectory and fly event parameters one hour before the launch.
- InterSV8-1: The control application should have the ability to provide the trajectory and fly event parameters one hour before the launch.

3. Step 3: 3D Relation Analysis

Once state variables are identified on subsystem level, the proposed state-based decomposition can be carried out. The decomposition follows the three dimensions mentioned above and defines relations between state variables. Three kinds of relations are defined:

- lifecyle relations: If a state variable has different usage states in different usage sub-phases, then each of them is defined as a lifecyle relation of the state variable.
- inter-subsystem relations: If a state variable has a relation with another subsystem, then we say that it has an inter-subsystem relation.
- internal-subsystem relations: If state variables can be further extracted out of a current state variable, then they are internal-subsystem relations of the state variable.

The relation decomposition is manually done by engineers, based on their experience. When we define state variables to represent these relations, we have the following rules:

(1) If a state variable has more than one lifecyle relation, a lifecycle relation is defined for each of the usage sub-phases respectively. For example, the state variable of ReqSV1, "The raw data process subsystem should have the ability to read received down-link data from different external hardware interface of the data receiver subsystem", has three usage states in the four defined utilization sub-phases:

- During the desktop-wired test sub-phase, the raw data processing subsystem should have the ability to read received down-link data from the USB device.
- During the desktop-wireless test sub-phase, the raw data processing subsystem should have the ability to read received down-link data from the wireless network card.
- During the pre-launch test sub-phase and the after-launch sub-phase, the raw data processing subsystem should have the ability to read received down-link data and the telemetry device states via serial communication protocol.

Although it has the same state in the pre-launch test sub-phase and the after-launch sub-phase, we define four seperate lifecycle relations for ReqSV1. we name them with a unique identifier number respectively: Phase-ReqSV1-1, Phase-ReqSV1-2, Phase-ReqSV1-3, and Phase-ReqSV1-4 in the sequence of the four defined usage sub-phases.

For the state variable of ReqSV8, "The trajectory and fly event parameters will not be defined until one hour before the launch", it is only required in one usage sub-phase:

- During the pre-launch test sub-phase, the raw data processing subsystem should have the ability to update trajectory and fly event data one hour before the launch.

Since the state is only required in the pre-launch test sub-phase, we define one lifecycle relation for ReqSV8 and give it a unique name PhaseSV8-3.

(2) If an inter-subsystem relation is identified together with a lifecycle relation, we say that the inter-subsystem relation is also usage sub-phase related. We define a separate inter-subsystem relation for each of them. For the ReqSV1 example, the following inter-subsystem interfaces are identified for the four usage sub-phases:

- the desktop-wired test sub-phase: the USB device interface with the data receiver subsystem.
- the desktop-wireless test sub-phase: the wireless network card interface with the data receiver subsystem.
- the pre-launch test sub-phase and the after-launch sub-phase: the serial communication interface with the data receiver subsystem.

Like its lifecycle relations, we define four inter-subsystem relations and name them Inter-ReqSV1-1, Inter-ReqSV1-2, Inter-ReqSV1-3, and Inter-ReqSV1-4 in accordance with the lifecycle relations.

4. Step 4: State Variable Assignment to each Usage Sub-phase

Based on the 3D relation analysis result, we assign state variables to the defined usage sub-phases. We follow the basic principles:

- For the lifecycle relations, the assignment is based on their usage sub-phases. For example, Phase-ReqSV1-1, Phase-ReqSV1-2, Phase-ReqSV1-3, and Phase-ReqSV1-4 are only applicable to the specific usage sub-phases. Therefore, as can be seen in Fig. 7, we assign Phase-ReqSV1-1 to the desktop-wired test sub-phase, Phase-ReqSV1-2 to the desktop-wireless test sub-phase, Phase-ReqSV1-3 to the pre-launch test sub-phase, and Phase-ReqSV1-4 to the after-launch sub-phase.
- For an inter-subsystem relation, if it is related with a lifecycle relation, then it is assigned to the corresponding usage sub-phase. For example, Inter-ReqSV1-1 is identified together with Phase-ReqSV1-1, therefore, as can be seen in Fig. 7, it is assigned to the same usage sub-phase as that of Phase-ReqSV1-1.
- If an inter-subsystem relation is only applicable to some specific usage sub-phases, it is assigned to these usage sub-phases respectively. For example, InterSV8-1, "The control application should have the ability to provide the trajectory and fly event data one hour before the launch", which represents the inter-subsystem relation between the raw data processing subsystem and the control application, is only required in the pre-launch test sub-phase. Therefore, as can be seen in Fig. 7, we assign InterSV8-1 to the pre-launch test sub-phase.
- If an inter-subsystem relation is applicable to all of the usage sub-phases, it is assigned to each of them. For example, Inter-ReqSV11-1, "the payload subsystem should define payload data format", which represents the inter-subsystem relation between the raw data processing subsystem and the payload subsystem, is required in all of the four defined usage sub-phases. Thus, as can be seen in Fig. 7, we assign it to each of the usage sub-phases.
- If a state variable has no lifecycle relations or inter-subsystem relations, it is assigned to each of the usage sub-phases. For example, ReqSV12, "The raw data processing subsystem should have the ability to process the data rate of 2M byte/s", has no lifecycle relations or inter-subsystem relations. In fact, it is a subsystem performance requirement. It should be considered on subsystem level. Thus, as can be seen in Fig. 7, we assign it to each of the usage sub-phases.

For the raw data processing subsystem example, the assignment of state variables to the four usage sub-phases is shown in Fig. 7.

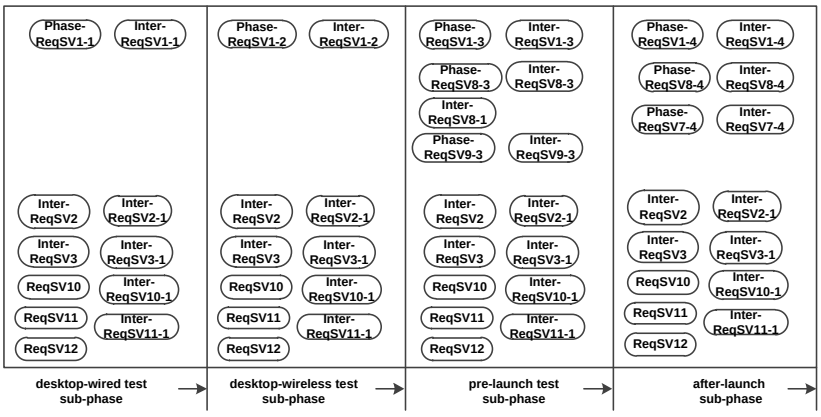

Figure 7. This figure shows state variable assignment of the raw data processing subsystem. It shows the different state requirements from its different usage sub-phases throughout the space mission lifecycle.

5. Step 5: Further Identification of State Variables for each State Variable

After state variables were decomposed and assigned on subsystem level, for each usage sub-phase, further identification of subsubsystem state variables will be extracted along the mentioned three dimensions. Steps

3-4 are repeated in order to complete the further decomposition. Again, this decomposition is carried out manually by engineers. We also have the following rules:

- In order to avoid the number of inter-subsubsytem relations as much as possible, we avoid merging the subsubsystem state variables decomposed from usage sub-phase-related state variables with those decomposed from usage sub-phase-independent state variables. For the raw data processing subsystem example, the state variable of Phase-ReqSV1-1 is usage sub-phase related, therefore, we keep its sub-subsystem state variable CSC1, "USB data reader component," independent of other subsubsystem state variables.

- We can directly reuse the subsubsystem state variables decomposed from usage sub-phase-independent state variables in all of the usage sub-phases. For the raw data processing subsystem example, the state variable of Inter-ReqSV2 is usage sub-phase independent, therefore, its subsubsystem state variable CSC2, "payload data processor component," is reused in all of the four usage sub-phases.

- If state variables represents non-function requirements on subsystem level, sometimes there is not one-to-one correspondence between components and the non-function requirement. We have to estimate it on subsystem level. Therefore, in the model, we directly relate it to all of the subsubsystem state variables. For the raw data processing subsystem example, its "RSR10: the data rate is 2M byte/s" is a subsystem level non-function requirement. When we design and implement the subsystem, we must keep in mind that it has effect on the data reader component, the data processing component, and the data display component. Therefore, all of the further decomposed subsubsystem state variables, i.e. CSC1, CSC2, and CSC3, are related to RSR10.

- After we got all of the subsubsystem state variables, a state variable assignment is carried out again. The purpose is to reuse the subsubsystem state variables as much as possible. Thereafter, a good design and implementation solution can be approximated. For the raw data processing subsystem example, after the assignment, some components such as "CSC2: payload data processor component" and "CSC3: payload data display component' are reused in all of the usage sub-phases. CSC1 is identified as an interface to the data receiver subsystem, therefore, it is an adapter and has different states in accordance with the usage sub-phases.

As mentioned, such decomposition work ends when current decomposition meets the requirements from the current development phase or programs can be implemented. Assuming the current development phase is at System Requirement Analysis phase, a static system model is required. To simplify the complexity, we only take the desktop-wired test sub-phase as an example. Its static system model is shown in Fig. 8.

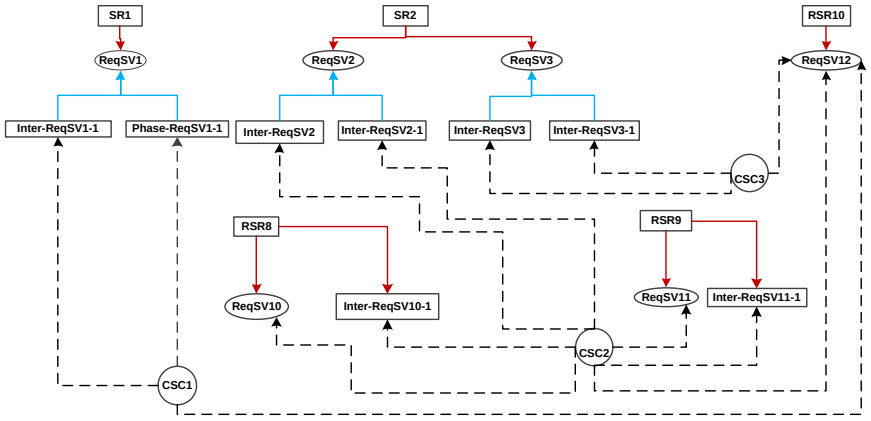

Figure 8. This is a SRR level static system model for the desktop-wired test sub-phase of the raw data processing subsystem. With the on-going development process, the model can be refined. All of the state variables in this model can be traced back to requirements.

As shown in Fig. 8, we can get the following information from the static system model:

- Three subsubsystem state variables are acquired: CSC1, CSC2, CSC3. CSC1 is from usage sub-phase-related state variables, so we keep it independent of CSC2 and CSC3. CSC2 and CSC3 are from usage sub-phase-independent state variables, so we can directly reuse them in the other usage sub-phases.

- Each state variable can be traced back to TRs. If a TR changes, we can immediately know the affected state variables. Since the implementation will be accomplished at the end of the development lifecycle and the model is refined accordingly, we can make a conclusion that every state variable can be directly traced back to TRs. A benefit from this is the automatic test case generation and execution. Furthermore, if there has been already a subsubsystem state variable whose TRs are exactly the same as CSC1, CSC2, or CSC3, we can directly reuse it because of the same TRs. With the development process goes on, the reuse possibility of other level model can also be evaluated. With such an advantage, we can implement the reuse of intermediate products on each level even the whole development lifecycle.

- "RSR10: the data rate is 2M byte/s" is a non-functional performance requirement. It is stated for the whole subsystem. All of the components should contribute to it. As can be seen in Fig. 8, RSR10 is directly related to CSC1, CSC2, and CSC3. Such correspondence helps to ensure that all of the TRs are exactly considered when we design and implement a subsystem.

C. Modeling Software Environment

One important aspect in implementing MBSE is to construct the model in a computer-aided environment. Therefore, we describe the modeling process with xCore.[15] Firstly, a meta model that represents the modeling process is generated. A tree view of the meta model can be seen in Fig. 9.

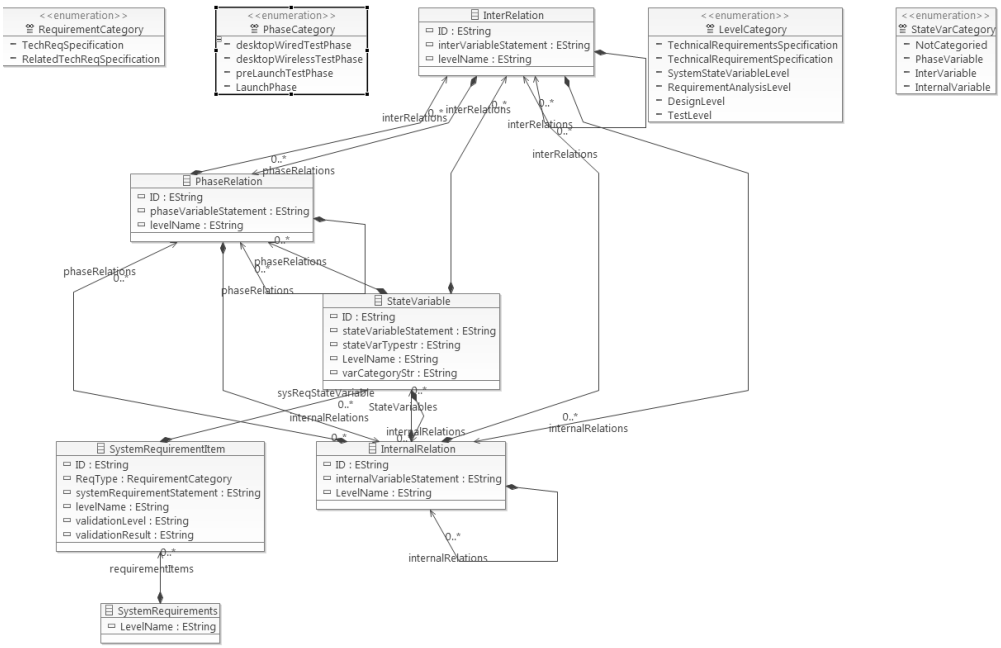

Figure 9. This is a tree view of the meta model representing the 3D state decomposition modeling process. With data defintions in the meta model, state-based decompostition can be performed in the Eclipse Modeling Framework (EMF).

As can be seen in Fig. 9, we can define the following items in the meta model:

- We can define TRs of a subsystem in the data type of "SystemRequirementItem". All of the instances of "SystemRequirementItem" are a list of TRs.

- We can define usage sub-phases in the data type of "PhaseCategory". It is based on usage state requirements throughout the lifecycle of a space mission.

- We can define development phases in the data type of "LevelCategory". Thereafter, we can specify the current level of a model or a state variable. For example, if the current model is a SRR level model, then "LevelCategory" has a value of "RequirementAnalysisLevel". With such a specification, we can label a model with the synchronized review points. If we want to reuse the intermediate products of a development process, we just need to evaluate the possibility to reuse the appropriate level model. For a state variable, if it has a "LevelCategory" value, then it is marked as the last decomposed state variable from its SRs or RSRs. For the raw data processing subsystem example, assuming that the current development phase is at System Requirement Analysis phase, CSC1, CSC2, and CSC3 are the last state variables of the current level model, then all of their "LevelCategory" are given a value of "RequirementAnalysisLevel".

- We can define state variables in the data type of "StateVariable". At the very first beginning, they are variables to represent states of TRs on subsystem level. With the decomposition work advances, they turn into parameters in programs. In any case, they can trace back to the TRs.

- We can define the three kinds of relations in the data type of "PhaseRelation", "InterRelation", and "InternalRelation". With these relations, the model can be decomposed level by level until the subsystem is implemented.

Since xCore is well integrated into Eclipse Modeling Framework (EMF),[16] which is a model-based framework for modeling and automatic code generation, engineers can use EMF's graphical modeling environment and manually construct a subsystem model. The model is stored as a ".xmi' file, a standardized format. Thereafter, model-based simulation as well as model-based verification and validation can be performed on top of the model-based framework.

VI. Conclusion

As the first step to achieve the goals of MBSE, a 3D state decomposition modeling method was proposed in this paper. With this method, information of a development process can be integrated into a model. Based on the model, we will conduct further research on the following topics:

- An analysis of the information that has to be integrated into the model. As described in this paper, the model in MBSE can be marked as a SRR level model, a PDR level model, a CDR level model, a QR level model, or an AR level model according to current review points. Therefore, in order to carry all of the information generated between these reviews, an analysis on what the information is and how to integrate it into the model will be studied.

- An analysis of what can exactly be reused in different development phases. As described in this paper, the intermediate development products could be reused on different levels. Therefore, an analysis on what these products are and how to extract them from the model will be carried out.

- Automatic documentation generation. As described in this paper, in traditional document-based engineering process, many documents have to be manually written and maintained, which is very time-consuming and error-prone. Since MBSE integrates all of the information of a development process into a model, it would be possible to automatically generate these documens. To make it a reality, model-based simulation and model-based verification and validation methods will be developed. By that time, the overheads caused by writing documentation will disappear and engineers can focus only on the modeling work.

- Automatic test case generation and execution. As described in this paper, the model starts from TRs and ends as programs. Therefore, a suite of test cases that can cover all of the TRs may be automatically generated, even executed. The feasibility of the automatic test case generation and execution will be demonstrated.

- Multidisciplinary analysis throughout the lifecycle of a space mission. As described in Section I, in MDO, it is impossible to implement a lifecycle analysis because the multidisciplinary design model can not trace back to requirements and the models for multidisciplinary analysis are not the same as the models used inside one single discipline. Now, with a lifecyle model for one discipline, plus distributed

simulation technology such as HLA and high performance computation technology, it is reasonable for us to conduct further research on a solution to perform consistent analysis and optimization of multidisciplinary design models throughout the whole lifecycle of a space mission.

However, time-based behavior of states has not been included in the model yet. All of these problems need to be studied and addressed in the future.

Our work is based on "Virtual Satellite" software framework, which leads to a novel solution to implement MBSE throughout the whole lifecycle of a space mission.

Acknowledgments

The authors would like to acknowledge the financial support from the China Scholarship Council (CSC) for this research.

References

[1] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules," August 2010, URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5553438 [cited 9 August 2012].

[2] Sobieszczanski-Sobieski, J. and Haftka, R. T., "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," *Structural Optimization*, Vol. 14, 1997, pp. 1-23.

[3] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification," August 2010, URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5557731&contentType=Standards [cited 9 August 2012].

[4] Reid, M., "An Evaluation of the High Level Architecture (HLA) as a Framework for NASA Modeling and Simulation," *Proceedings of the 25th NASA Software Engineering Workshop, Goddard Space Flight Center*, Maryland, November 2000.

[5] Agte, J., deWeck, O., Sobieszczanski-Sobieski, J., Arendsen, P., Morris, A., and Spieck, M., "MDO: Assessment and Direction for Advancement - an Opinion of One International Group," *Structural and Multidiscipilnary Optimization*, Vol. 40(1), 2010, pp. 17–33.

[6] "ECSS-M-ST-10C Rev.1 - Project planning and implementation," URL: http://www.ecss.nl [cited 20 July 2012].

[7] "ECSS-E-ST-10-06C - Technical requirements specification," URL: http://www.ecss.nl [cited 20 July 2012].

[8] "ECSS-E-ST-40C Space Engineering - Software," URL: http://www.ecss.nl [cited 20 July 2012].

[9] Jansma, P. A. T. and Jones, R. M., "Advancing the Practice of Systems Engineering at JPL," *IEEE Aerospace Conference*, Big Sky, Montana, USA, March 4-11 2006.

[10] Schaus, V., Fischer, P. M., Luedtke, D., Braukhane, A., Romberg, O., and Gerndt, A., "Concurrent Engineering Software Development at German Aerospace Center - Status and Outlook," *Proceedings of the 4th International Workshop on System and Concurrent Engineering for Space Applications, European Space Agency (ESA)*, 2010.

[11] "ECSS-E-TM-10-25A Space Engineering-Engineering Design Model Data Exchange (CDF)," URL: http://atlas.estec.esa.int/uci_wiki/tiki-index.php [cited 20 July 2012].

[12] "ECSS-E-TM-10-23A Space Engineering-Space System Data Repository," URL: http://atlas.estec.esa.int/uci_wiki/tiki-index.php [cited 20 July 2012].

[13] Ingham, M., Rasmussen, R., Bennett, M., and Moncada, A., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *AIAA Intelligent Systems Technical Conference*, AIAA, Chicago, IL, USA, September 20-22 2004.

[14] Rasmussen, R., Ingham, M., and Dvorak, D., "Achieving Control and Interoperability through Unified Model-based Systems and Software Engineering," *AIAA Infotech at Aerospace 2005, Arlington, VA*, Arlington, VA, USA, September 26-29 2005.

[15] URL: http://wiki.eclipse.org/Xcore [cited 20 July 2012].

[16] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E., *EMF: Eclipse Modeling Framework, 2nd ed*, Addison-Wesley, 2009.