IAC-12-D1.3.7

STATIC SIMULATION SCHEDULING FOR THE VALIDATION OF SPACE SYSTEM REQUIREMENT DECOMPOSITION

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

Andreas Gerndt[*], **Bo Liu**[t]

One important issue in space engineering is to ensure that all system requirements are always fulfilled throughout the whole development process. To perform this task, various validation technologies have been developed. In this paper, document-based reviews and Modelling & Simulation technology are illustrated as two of such technology examples. Their disadvantages are analysed as well. The fundamental challenge for them is to carry out a consistent validation throughout the whole development process. To resolve this problem, a static simulation scheduling approach is proposed in this paper. This scheduling approach is based on a hierarchical decomposition model of a space system. The model is constructed following the principles of Model-Based Systems Engineering (MBSE). It maps all requirements to state variables and defines states as well as relations along three red lines of a space mission: inter-subsystems, internal subsystems, and space mission lifecycle. In the static simulation scheduling approach, the validation is driven just by the states of state variables and their relations. But, the specific validation process is dependent on current states of the processed variable. The scheduling approach is illustrated and demonstrated in this paper. It facilitates a preliminary validation analysis throughout the whole development lifecycle that would otherwise be cost intensive or even be impossible. Furthermore, after each system refinement, the validation analysis can be performed again automatically. Our work is based on the "Virtual Satellite", which is being developed by German Aerospace Center (DLR). We are pursuing the possibility to implement specification, analysis, design, and operation of space missions in a unified framework.

I. INTRODUCTION

Space system products, hardware or software, have great demands on quality assurances. One of those is to ensure that all system requirements are fulfilled when the products are finally delivered. Requirements are the baseline from which a space system can be developed. In space engineering, the development process of requirements to final products is divided into several phases [1]. Normally, in order to ensure that specified objectives are accomplished in these phases, proof has to be provided to validate that the current development phase is going well. To present such proof, various techniques, e.g. "review, inspection, testing, walk-through, cross-reading, desk-checking, model simulation, and many types of analysis such as traceability analysis, formal proof or fault tree analysis" [2] are specified. In space engineering, these techniques are also used to validate whether system requirements are always fulfilled throughout the whole development process.

In this paper, firstly, in Section II, we will present two kinds of such techniques, which are currently being used in space engineering. Although effective, these techniques have some shortcomings. A static simulation scheduling that can also validate the fulfilment of requirements will be proposed in this paper. This approach is based on a hierarchical model of a space system. The key features of the model will be described in Section III. In Section IV, the approach will be illustrated in detail. To demonstrate its feasibility, we examine a subsystem in a sounding rocket based exploration system as an example. The validation result will be presented in Section V. Section VI closes this paper with a conclusion and the direction for future work.

II. CURRENT VALIDATION APPROACHES

II.I Modelling & Simulation

Modelling & Simulation can be used to support system level validation and operation activities [3]. By means of Modelling & Simulation technology, a model of the system-to-be is constructed and analysis results

[*] German Aerospace Center (DLR), Simulation and Software Technology, Braunschweig, Germany,
Email: Andreas.Gerndt@dlr.de
[t] Center for Space Science and Applied Research, Chinese Academy of Sciences, Beijing, China,
Email: boliu@nssc.ac.cn

can be obtained on top of the model. Evaluation whether system requirements are met or not can then be carried out. "Use of Modelling & Simulation in a coherent way at system-level across the lifecycle of a project potentially yields significant benefits to a project." [3]

Many technologies which support dynamical simulation have been developed, e.g. High Level Architecture (HLA) [4] [5] and Simulation Model Portability (SMP) [6]. We consider the HLA as an example that "provides a general framework within which simulation developers can structure and describe their simulation applications." Relying on the HLA, a simulation environment can be created and simulation can be carried out in it. However, in order to construct an executable HLA-based simulation, the following work has to be done in advance.

- Data definitions of interfaces among simulation applications have to be defined in a simulation object model (SOM) or a federation object model (FOM) [5].
- Internal processes of each interface have to be specified and implemented in the individual simulation application.
- Time-based transition processes of interfaces and internal processes have to be defined and implemented as well.

On the other hand, the development process of a space system has the following characteristics:

- Subsystems are highly related with each other. These relations vary from phase to phase throughout the whole development process.
- Granularity of requirement decomposition changes throughout the whole development process. At the very beginning, only requirements are specified. Since requirements are proposed mainly by customers, they are abstract and not precise enough to support the development of products. Therefore, requirements need to be decomposed into smaller and more concrete elements. For hardware, requirements are decomposed into parts with interface definition in-between until final products can be manufactured and assembled. For software, requirements are decomposed into components until programs can be implemented, also with interface definition in-between. With the decomposition goes on, the internal processes of interfaces have to change accordingly.

To adapt to these characteristics, the HLA-based simulation environment has to change accordingly every time when some change happens in the development process. However, when the change happens, the whole simulation environment has to be debugged and tested again before it can be used to provide new validation results. Therefore, the main challenge for an environment which supports dynamic simulation is the inevitable changes throughout the development process of a space system versus the difficulty to adapt to these changes.

In a word, although it is essential to use "system simulation through system level requirement definition, analysis and design trade-offs, finally to training and support for operations" as a coherent approach, it is still very difficult to implement a consistent validation analysis by means of these simulation technologies.

II.II Document-based Reviews

The whole development process is divided into Preliminary Definition phase, Detailed Definition phase, and Qualification & Production phase. As shown in Fig. 1, reviews are specified in-between like the System Requirements Review (SRR), the Preliminary Design Review (PDR), Critical Design Review (CDR), Qualification Review (QR), and the Acceptance Review (AR) [1].

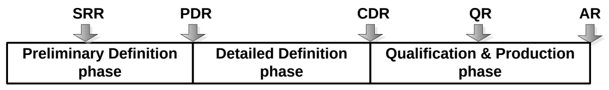
SRR
PDR
CDR
QR
AR
Preliminary Definition phase
Detailed Definition phase
Qualification & Production phase

Fig. 1: This figure shows a development process, which is divided into phases. Reviews are set for these phases. The purpose of these reviews includes validating that all requirements are fulfilled.

For each review, one important objective is to ensure that all requirements are still fulfilled. Taking the development process of a space software product as an example, various traceability matrices are defined to describe the traceability relationship between requirements and lower-level components [7]:

- For the PDR, a traceability matrix between software architectural design and requirements are defined. It is used to describe "the traceability between the requirements and the software components".
- For the CDR, a traceability matrix between architectural design and detailed design is defined. It is used to describe "the traceability between the architecture and the detailed design".
- For the CDR, a traceability matrix between software code and requirements is also defined. It is used to ensure that "the code is traceable to design and requirements".
- For the CDR, a traceability matrix between software unit tests and requirements is defined. It is used to ensure that "the unit tests are traceable to software requirements, design, and code".

These traceability matrices are necessary items for the individual reviews mentioned above. In document-based space engineering, engineers have to manually fill in a "Requirement Traceability Matrix (RTM)" table, in which the above traceability matrices can be illustrated.

Actually, the whole development process involves different engineers. Taking a space software product as an example, an architecture design solution is proposed by designers, whereas, programs are developed by programmers. Information transfer between them is by means of a large number of documents, e.g. software requirements specification, software interface control documents etc. Every time when engineers fill in the RTM table, they have to look into all of the related documents. Thereafter, they submit it as one part of documents to the review board. On the other side, in order to check the validity and correctness of a RTM table, reviewers also have to investigate all these documents again.

However, due to different personal experience, engineers may have different understanding of the same document and the same RTM table, so do reviewers. Thus, such a document-based approach is heavily dependent on personal experience and bears the risk of inconsistency when assessment is made. Furthermore, any changes such as a requirement change or a design modification require re-reviews of these documents again, which causes heavy workload both for engineers and reviewers.

In order to eliminate overheads generated by the above document-based approach, a static simulation scheduling approach will be presented in the following sections. With this approach, validation analysis of the fulfilment of requirements can be automatically generated and evaluated. Both engineers and reviewers need not look into so many documents any more. This approach is based on a hierarchical decomposition model of a space system. The key features of the model will be illustrated in Section III.

III. KEY FEATURES OF THE HIERARCHICAL DECOMPOSITION MODEL

The hierarchical decomposition model is constructed following the principles of Model-Based Systems Engineering (MBSE) [9]. It can be used throughout the whole space mission lifecycle.

As shown in Fig. 2, a typical lifecycle of a space mission is divided into seven phases: Phase 0, Phase A, Phase B, Phase C, Phase D, Phase E, and Phase F [1]. Phase B, together with Phase C and Phase D, is defined as a subsystem development lifecycle. After the subsystem is delivered, it commences Phase E. Phase E describes operations/utilization states on system level. The disposal plan is implemented in Phase F [1].

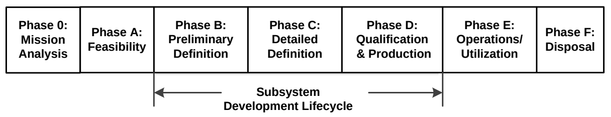
Phase 0: Mission Analysis
Phase A: Feasibility
Phase B: Preliminary Definition
Phase C: Detailed Definition
Phase D: Qualification & Production
Phase E: Operations/ Utilization
Phase F: Disposal
Subsystem Development Lifecycle

Fig. 2: This figure shows a typical lifecycle of a space mission.

The model is to carry all information generated throughout the space mission lifecycle. It has the following key features [10]:

Firstly, the model is based on technical requirements (TRs). TRs are established at the end of Phase A and assigned to subsystems. They "exclude requirements such as cost, methods of payment, quantity, time or place of delivery" [11]. In the model, each TR is translated into variables describing certain subsystem states.

Secondly, state variables are further decomposed into states. States can be either properties of state variables or relations in-between. For each state variable, the state-based decomposition continues until final products can be implemented. In the model, three types of relations are considered:

- Lifecycle relations: For a state variable, lifecycle relations describe its different states in different usage sub-phases. Usage sub-phases represent the ways to use a subsystem or a component throughout the development lifecycle, Phase E, and Phase F.
- Inter-subsystem relations: Inter-subsystem relations describe the dependency between the subsystem and other subsystems. If a state variable represents a relation with another subsystem, we say it has an inter-subsystem relation.
- Internal-subsystem relations: Internal relations are further extracted out of a current state variable.

Thirdly, the model carries all information generated from TRs to current development phase. As shown in Fig. 3, in model-based engineering process, a System Requirement (SR) phase model, a Preliminary Definition (PD) phase model, a Detailed Definition (DD) phase model, and a Qualification & Production (QP) phase model can be defined in accordance with the defined development phases [10].

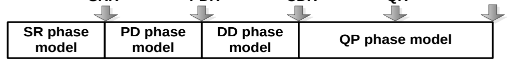

Fig. 3: This figure shows a model-based engineering process. The individual phase model represents all information generated in the decomposition process from TRs to current development phase.

To illustrate the model, we examine a raw data processing subsystem of a sounding rocket based exploration mission as an example [10]. The example will also be used to demonstrate the feasibility of the proposed static simulation scheduling approach.

ID	TR Statement
TR1	The raw data processing subsystem should read received down-link data from the data receiver subsystem.
TR2	The raw data processing subsystem should unpack, process, and display payload data.
TR3	The raw data processing subsystem should generate raw data packages for the science data processing subsystem.
TR4	The raw data processing subsystem should display the trajectory and fly events after the sounding rocket is launched.
TR5	The raw data processing subsystem should monitor the telemetry device states.
TR6	The trajectory and fly event parameters will not be defined until one hour before the launch.
TR7	There is only one telemetry device in this mission.
TR8	Payload data package should meet CCSDS standard.
TR9	There are three payloads in the mission: payload A, B, C.
TR10	Down-link data rate is 2M byte/s.

Table 1: This is a TR list for the raw data processing subsystem. Each TR is assigned a unique identifier number.

The sounding rocket based exploration mission is comprised of a raw data processing subsystem, a sounding rocket subsystem, a payload subsystem, an on-board data management subsystem, a data receiver subsystem, a science data processing subsystem, and a control application etc. These subsystems together are supposed to fulfil all system requirements of the space mission. For the raw data processing subsystem example, its TRs are listed in Table 1, each TR with a unique identifier number.

Four usage sub-phases was identified for the raw data processing subsystem: a desktop-wired test sub-phase, a desktop-wireless test sub-phase, a pre-launch test sub-phase, and an after-launch sub-phase [10]. Assuming we are currently in the System Requirement phase, a static system model is required.

Fig. 4 shows a tree view of the static system model. We can see that the model starts from TRs of the raw data processing subsystem. TRs are mapped to Requirement State Variables (ReqSVs), or Inter-subsystem State Variables (InterSVs), or both. A ReqSV indicates a state that can represent a TR, whereas an InterSV represents a relation between the subsystem and another subsystem. The relations of ReqSVs and InterSVs are further decomposed along the three defined dimensions. For example, ReqSV1 has different state requirements in the four defined usage sub-phases. Therefore, it has four lifecycle relations: Phase-ReqSV1-1, Phase-ReqSV1-2, Phase-ReqSV1-3, and Phase-ReqSV1-4. The Phase-ReqSV1-1 example has an inter-subsystem relation depicted as Inter-ReqSV1-1, which represents a relation between the raw data processing subsystem and the data receiver subsystem. CSC1, which is a computer software component (CSC), is an internal-subsystem relation of the Phase-ReqSV1-1 example. More details about the model construction are given in [10].

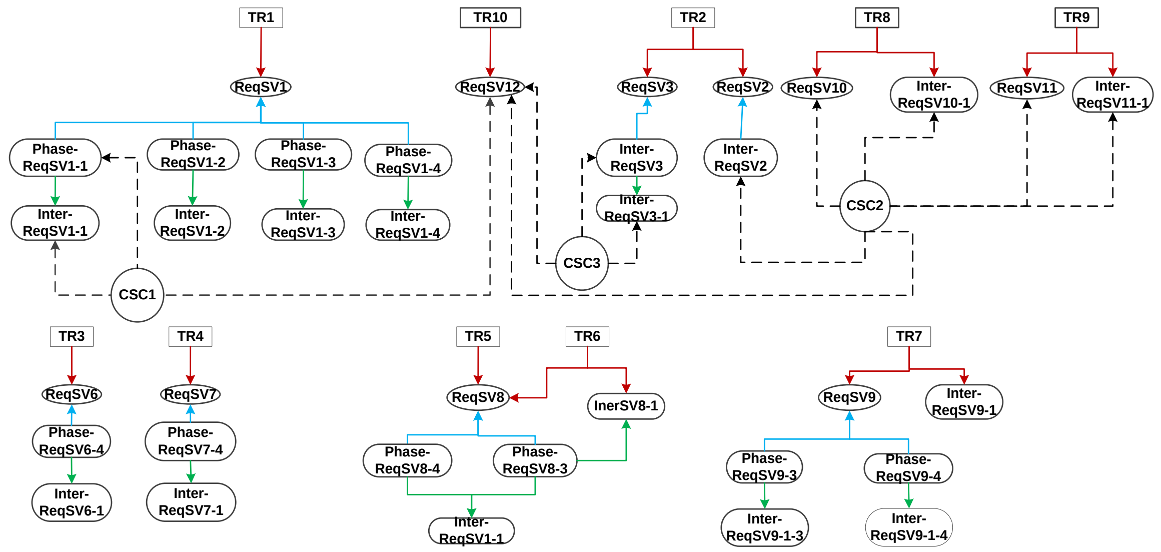

Fig. 4: This is a tree view of a static system model of the raw data processing subsystem. It is created in the Requirement Analysis phase.

IV. STATIC SIMULATION SCHEDULING FOR VALIDATION

Since the model carries all information of the subsystem, model-based technologies can be developed to conduct various validation analyses. The static simulation scheduling approach presented in Section IV is one of such technologies. In comparison to the simulation technologies described in Section II, it is a simplified analysis which does not consider interface data types among subsystems, transition processes of their relationships, and internal processes. Its purpose is to automatically conduct a preliminary validation analysis of the fulfilment of TRs in the current phase mode

IV.I Validation Criterion

Basically, to validate whether a TR is fulfilled or not, the criterion is that "each TR should be linked to at least one element of a lower level" [2]. In model-based space engineering, this means that there should be at least one lower-level component that can be traced back to the TR.

In our static simulation scheduling approach, the validation criterion is dependent on the characteristics of a specific TR and its state variables.

If a TR is a functional requirement, which defines what the subsystem shall perform [11], it has a one-to-one or one-to-many correspondence between it and lower-level components. This means that more than one component can be traced back to the TR. In the proposed static simulation approach, starting from the TR, if it can reach at least one lower-level component, we say that the TR is satisfied. For example, although TR2 "the raw data processing subsystem should unpack, process, and display payload data" can be traced to both CSC2 and CSC3, we can say TR2 is fulfilled after the traceability relationship between it and CSC2 or CSC3 is found, without the necessity to find both CSC1 and CSC2.

If a TR is a requirement that constrains the whole subsystem, all lower-level components have to contribute to it. This means that all lower-level components should be traced back to it. In our static simulation approach, starting from the TR, if it can reach all lower-level components, we say that the TR is satisfied. In the raw data processing subsystem example, TR10 "down-link data rate is 2M byte/s" is a subsystem-level non-functional performance requirement, therefore, the implementation of all lower-level components has to take it into consideration. All components, i.e. CSC1, CSC2, and CSC3, are traced back to it, as shown in Fig. 4. Therefore, we say that TR10 is fulfilled in the model.

For a state variable, if it has more than one lifecycle relations, each of these relations has to be covered by lower-level components. For example, in Fig. 4, ReqSV1 has four lifecycle relations. This means, we can say ReqSV1 is fulfilled only if all of the four lifecycle relations can reach at least one component respectively. However, in Fig. 4, only Phase-ReqSV1-1 can reach a lower-level component, i.e. CSC1. There is no other correspondence between lower-level components and Phase-ReqSV1-2, Phase-ReqSV1-3, or Phase-ReqSV1-4. Therefore, we conclude that ReqSV1 has not been fulfilled in the model yet.

IV.II Validation-related Properties or Parameters

As mentioned, our purpose is to perform the validation analysis automatically. Therefore, we define the following properties or parameters, so that the above validation criterion can be executed on top of the model.

- The "SubsystemLevelReq" property: For TRs, we define this property to recognize whether a TR affects the whole subsystem or not. For those who affect the whole subsystem, we assign it a value of "Yes". For those who only have an effect on some of the lower-level components, e.g. functional requirements, we assign it a value of "No".
- The "CurrentPhase" property: If a state variable represents the last component that can be decomposed from a TR in the current development phase, we assign it a value. Otherwise, it has no value. The specific value is dependent on the current development phase. The possible values are listed in Table 2. For example, if a state variable is the last node of a TR in System Requirement phase, its "CurrentPhase" property has a value of "SRPhase".
- The "ValidationPhase" property: This property defines the ultimate phase for a TR to be validated. Its value can be "SRPhase", "PDPhase", "DDPhase", and "QPPhase", in accordance with the phases defined in the whole development process. For example, if the "ValidationPhase" property of a TR has a value of "SRPhase", then the TR can be validated at the end of the System Requirement phase. If its value is "PDPhase", then the TR can only be validated at the end of System Requirement phase and at the end of Preliminary Design phase. After Preliminary Design phase, it cannot be validated any more.

Current Development Phase	**Correspondence Value of "CurrentLevel"**
System Requirement phase	SRPhase
Preliminary Definition phase	PDPhase
Detailed Definition phase	DDPhase
Qualification & Production phase	QPPhase

Table 2: This table shows the correspondence between current development phase and the value of the "CurrentPhase" property.

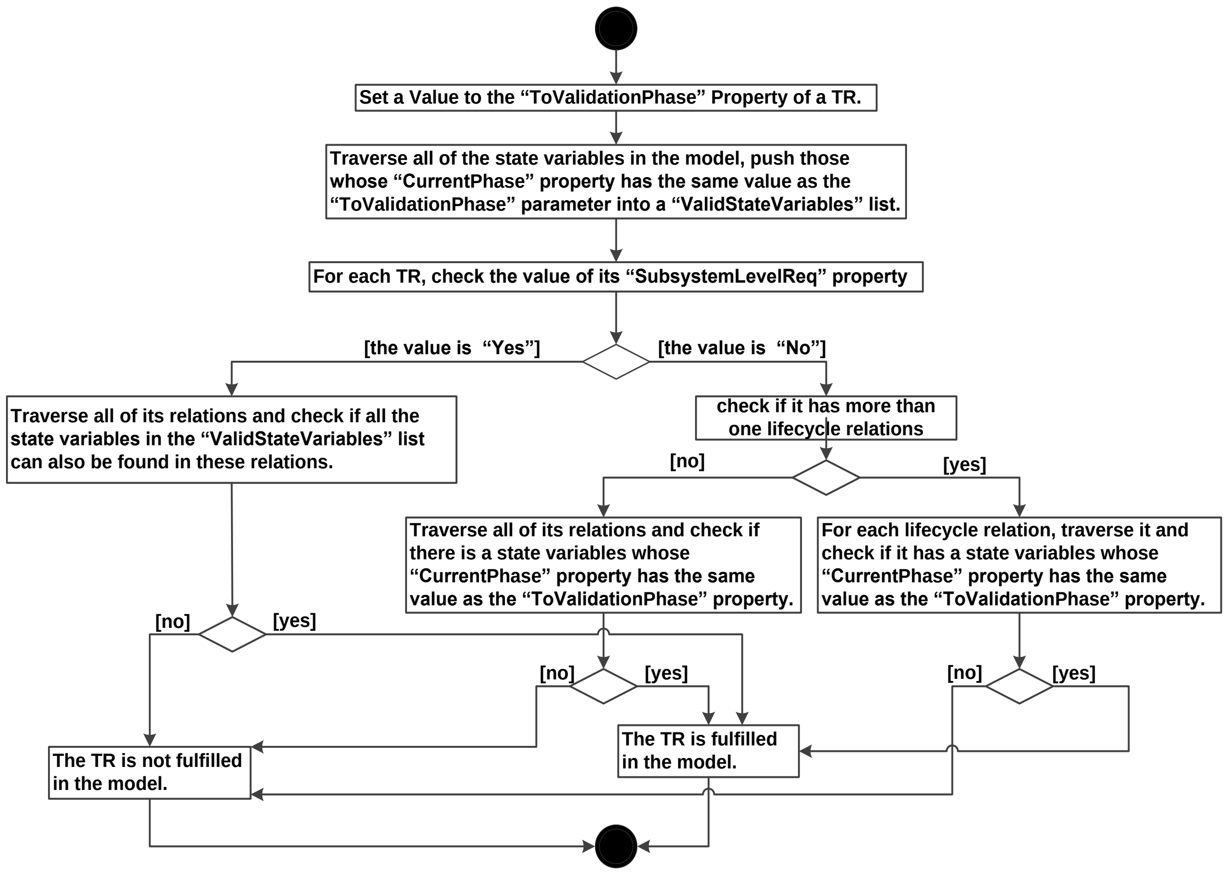
Set a Value to the "ToValidationPhase" Property of a TR.
Traverse all of the state variables in the model, push those whose "CurrentPhase" property has the same value as the "ToValidationPhase" parameter into a "ValidStateVariables" list.
For each TR, check the value of its "SubsystemLevelReq" property
[the value is "Yes"]
[the value is "No"]
Traverse all of its relations and check if all the state variables in the "ValidStateVariables" list can also be found in these relations.
check if it has more than one lifecycle relations
[no]
[yes]
Traverse all of its relations and check if there is a state variables whose "CurrentPhase" property has the same value as the "ToValidationPhase" property.
For each lifecycle relation, traverse it and check if it has a state variables whose "CurrentPhase" property has the same value as the "ToValidationPhase" property.
[no]
[yes]
[no]
[yes]
[no]
[yes]
The TR is fulfilled in the model.
The TR is not fulfilled in the model.

Fig. 5: This figure shows the validation process. The process is driven just by states of state variables and their relations. Different property values lead to different process branches.

There is also a "ToValidationPhase" parameter. This parameter defines the phase that is going to perform the validation. For example, if the "ToValidationPhase" property of a TR has a value of "SRPhase", it means the validation will be carried out at the end of System Requirement phase.

IV.III Validation Process

In our approach, all state variables are treated uniformly. However, the validation process of a TR is dependent on states of state variables and their relations mentioned above. Fig. 5 shows the validation steps.

Step 1: Firstly, a specific value is assigned to the "ToValidationPhase" parameter. As described, the value can be "SRPhase", "PDPhase", "DDPhase", and "QPPhase", in accordance with the current development phase. If the development phase it represents is after that of the "ValidationPhase" property, the validation is invalid. As mentioned, the "ValidationPhase" property represents the ultimate development phase that could be possible to perform the validation. We cannot validate whether it is fulfilled or not any more after this development phase. If a TR's "ToValidationPhase" property has a value of "PDPhase", for instance, but its "ValidationPhase" has a value of "SRPhase", the validation will be stopped immediately.

Step 2: All state variables in the model are traversed. If the value of the "CurrentPhase" property is the same as that of the "ToValidationPhase" property, it will be added to a "ValidStateVariables" list. This list will be used as one basis to validate subsystem-level TRs.

Step 3: The value of "SubsystemLevelReq" property is checked. If the value is "Yes", it means that the TR affects all components. Then, all of its relations will be traversed to check whether it contains all state variables in the "ValidStateVariables" list. If it does, we say it is fulfilled in the model. Otherwise, it is not fulfilled yet.

Step 4: If a TR's "SubsystemLevelReq" property has a value of "No", then the number of its lifecycle relations will be checked. If it has more than one lifecycle relations, then all of these lifecycle relations will be traversed. If all of them have at least one state variable whose "CurrentPhase" property has the same value as that of the "ToValidationPhase" property, we say the TR is fulfilled. Otherwise, it is not fulfilled in the model. If the number of lifecycle relations is only one, the TR's relations will also be traversed. If there is a state variable whose "CurrentPhase" value is the same as that of the "ToValidationPhase" property, we say the TR is fulfilled in the model. Otherwise, the TR is not fulfilled.

The above steps are the validation process for one TR. If we are to validate whether all TRs are fulfilled or not, we only need to repeat these steps.

IV. IV Implementation of the Validation Approach

Fig. 6 illustrates the main activities related with the implementation of the proposed approach. The activities at the specification level are independent of implementation codes and are performed by domain engineers.

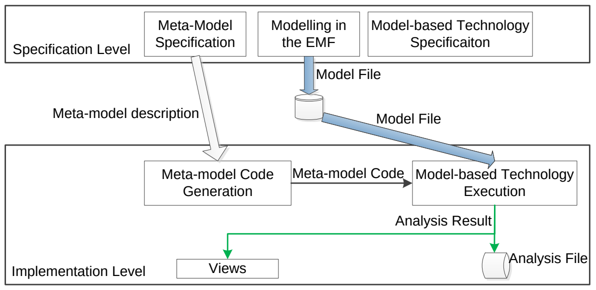

Fig. 6: This figure shows a data flow diagram among main activities to conduct a model-based engineering process in EMF. The activities at the specification level are carried out by domain engineers. The activities at the implementation level are carried out by software engineers or by the EMF automatically.

Meta-Model Specification: As mentioned, the proposed static simulation scheduling approach is based on a model. The process to construct the model is described as a meta-model with xCore [10] [13]. A meta-model is a general model from which models can be defined [12].

Modelling in the EMF: Based on the specified meta-model, domain engineers can manually construct the subsystem model in the graphical modelling environment of EMF. The model is then stored as an ".xmi" model file, which is used to perform model-based simulation.

Model-based Technology Specification: Model-based technologies are specified to carry out various validation analyses, e.g. to validate the fulfilment of requirements. Domain engineers, reviewers, even non-technical stakeholders can propose their own validation analysis requirements and model-based technology can be specified accordingly, like we already did for the proposed static simulation scheduling approach.

At the implementation level, the work done at the specification level is translated into software programs. Some code is automatically generated and some has to be developed by software engineers.

Meta-model Code Generation: After the specification of a meta-model, the correspondence meta-model code is automatically generated by EMF. With the code, instances of the meta-model can be created and states defined in the meta-model can be accessed via programming language.

Model-based Technology Implementation: Based on the meta-model code, model-based technology can be executed after it is implemented as software programs. For the proposed static simulation scheduling approach example, it has been specified in previous Sections. Now, we only need to implement it as software plug-in in the EMF. Thereafter, whenever a validation analysis needs to be performed, the approach can be automatically executed and analysis results are displayed or stored.

Validation Result	**ID**	**TR Statement**
false	TR1	The raw data processing subsystem should read received down-link data from the data receiver subsystem.
true	TR2	The raw data processing subsystem should unpack, process, and display payload data.
false	TR3	The raw data processing subsystem should generate raw data packages for the science data processing subsystem.
false	TR4	The raw data processing subsystem should display the trajectory and fly events after the sounding rocket is launched.
false	TR5	The raw data processing subsystem should monitor the telemetry device states.
false	TR6	The trajectory and fly event parameters will not be defined until one hour before the launch.
false	TR7	There is only one telemetry device in this mission.
true	TR8	Payload data package should meet CCSDS standard.
true	TR9	There are three payloads in the mission: payload A, B, C.
true	TR10	Down-link data rate is 2M byte/s.

Table 3: This table shows the validation results of the static system model of the raw data processing subsystem.

V. VALIDATION EXAMPLE

We use the static system model of the raw data processing subsystem shown in Fig. 4 to demonstrate the feasibility of our proposed approach. As mentioned, the model is developed in the Requirement Analysis phase. The "ToValidationPhase" parameter is assigned a value of "SRPhase". CSC1, CSC2, and CSC3 are assigned a value of "SRPhase" for their "CurrentPhase" property respectively. TR10 is assigned a value of "Yes" to its "SubsystemLevelReq" property. We input all the information into the model and trigger the execution of the static simulation scheduling approach. The validation result is shown in a table view, as expected.

As shown in Table 3, TR2, TR8, TR9, and TR10 are considered to be already fulfilled in the decomposition model. TR1 has four lifecycle relations and only one of them is linked to CSC1, therefore, its validation result is "false". Since TR3, TR4, TR5, TR6, and TR7 cannot reach any of the lower-level components, their validation result is "false" accordingly.

With the development process advances, the decomposition of requirements advances as well. Every time when it is necessary to carry out such a validation task, we just repeat the same work and validation result can be automatically obtained in a consistent way. The validation results should be fed back into the development process and support activities such as reviews.

VI. CONCLUSION

This paper proposed a static simulation scheduling approach which can be used to validate whether requirements are fulfilled or not throughout the whole decomposition process. Although it can only generate preliminary analysis results, it can be consistently used throughout the whole development lifecycle, which is very difficult and costly for some current validation techniques.

This approach is based on a model which is constructed following the principles of MBSE. Therefore, the advantages of this approach also demonstrate that MBSE has great potential application in space engineering.

Our work is based on the "Virtual Satellite", developed by German Aerospace Center (DLR) [14]. This framework already handles the specification of the meta-model and offers a variety of views to analyse the results. Thus, we only need to concentrate on the specification and implementation of the static simulation scheduling approach itself.

However, besides static analysis, there are other important validation requirements, e.g. time-based validation and run-time error detection. In order to provide such validation results more precisely, we will conduct research to resolve these problems.

ACKNOWLEDGMENTS

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

REFERENCES

[1] ECSS-M-ST-10C, Space Project Management - Project Planning and Implementation, March 2009, URL: http://www.ecss.nl [cited 20 July 2012].

[2] ECSS-Q-ST-80C, Space Product Assurance - Software Product Assurance, March 6, 2009, http://www.ecss.nl/ [cited on 25 July 2012]

[3] ECSS-E-TM-10-21A, Space Engineering - System Modeling and Simulation, April 2010, URL: http://www.ecss.nl [cited 20 July 2012].

[4] 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].

[5] 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].

[6] ESA, SMP 2.0 Handbook, Issue 1, Revision 2, 2005, doc Ref: EGOS-SIM-GEN-TN-0099.

[7] ECSS-E-ST-40C, Space Engineering - Software, March 6, 2009, http://www.ecss.nl/ [cited on 25 July 2012]

[8] ECSS-E-ST-10C, Space Engineering - System Engineering General Requirements, March 6, 2009, http://www.ecss.nl/ [cited on 25 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] Hao Zhang, Andreas Gerndt, and Bo Liu, 3D State Decomposition Modeling Method for MBSE in Space Engineering, AIAA SPACE 2012 Conference & Exposition, Pasadena, CA, September 11 - 13, 2012.

[11] ECSS-M-ST-10-06C, Space engineering-Technical requirements specification, March 2009, URL: http://www.ecss.nl [cited 20 July 2012].

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

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

[14] V. Schaus, P. M. Fischer, D. Lüdtke, A. Braukhane, O. Romberg, and A. Gerndt, Concurrent engineering software development at German Aerospace Center - status and outlook, the 4th International Workshop on System and Concurrent Engineering for Space Applications, European Space Agency (ESA), 2010.