Formal modelling language to simulate distributed autonomous on-board system

Attila Baksa$^{(1)}$, Balint Sodor$^{(1)}$, Gabor Troznai$^{(1)}$, Dr. Sandor Szalai$^{(1)}$

$^{(1)}$SGF Ltd.
Pipiske u. 1-5/20, 1121 Budapest, HUNGARY

ABSTRACT

Applying formalized behavioural models for interface level simulation of distributed and autonomous aerospace systems during the whole development life-cycle may play a key role in increasing robustness and decreasing costs. In system engineering and development the cost of correcting failures is increasing heavily as the development progresses. Modelling and simulation are essential tools for unfolding either design or implementation failures in the early phase of the development even when the components of the on-board system are not available. Utilizing formal methods in modelling and simulation provides mathematically based techniques for the specification, development and verification of on-board systems.

Especially in large scale, science related space missions the on-board data handling and processing logic is often implemented by a number of autonomous embedded modules. In general these on-board modules are developed and implemented in parallel by a number of different teams. To reduce development cost and increase the robustness of the whole on-board system the behaviour of it should be evaluated long time before the integration phase of the system development. In later phases the formalized models of the system modules serves as a basis of hardware-in-the-loop simulation for validation and verification.

This paper presents our concepts on a semantically anchored domain specific modelling language. The elaborated language can be used to perform behaviour simulation of on-board systems used in aerospace. The modelling language provides tools for state based modelling of on-board equipments either in nominal and non-nominal states. As the target of the modelling language is the high reliability mission critical system domain it is inevitable to investigate the system behaviour in both nominal and non-nominal operation states. For verification of fault tolerance of a subsystem the modelling language provides fault injection services as well. Fault injection and automated processes can be used to verify the autonomous behaviour of the entire system as well. A case study on the elaborated modelling and simulation tools is presented in this article as well.

INTRODUCTION

Onboard system development process in the field of space missions – especially in case of science related missions – often has the following specific characteristics compared to the commercial industrial embedded system development:

- Long term development lifecycle
  - Usually long planning and specification phase
  - Usually long system development phase
- Usually long flight/operation phase – where software development and operation cyclogram development may take place as well
- Highly parallel development due to the distributed nature of the onboard systems. Onboard system usually built up by a number of autonomous loosely coupled embedded systems (subsystems and scientific units) connected to a well-known onboard bus system.

In the Wigner RCP of the Hungarian Academy of Sciences and the SGF ltd. we have spent decades developing Electrical Ground Support Equipments (EGSEs) simulating onboard electrical interfaces for different space missions [3] and onboard spacecraft systems[1, 2]. For the European Space Agency's (ESA) Rosetta mission we have implemented the simulator of the Philae lander's onboard system (called Lander Software Simulator - LSS). This simulator uses a dedicated modelling framework and a special hardware module to perform signal level simulation of the onboard equipment.
Based on our experience we have defined a system class for the onboard systems used in space missions. A spacecraft contains a number of scientific instruments and subsystems (units) communicating with a central control and acquisition

unit. These onboard units usually contain some internal processing and communication logic implemented on a dedicated processor card or FPGA built up by space classified hardware elements. The maintainability and reparability of onboard systems are strongly restricted during operation phase only the software components can be updated partly. Due to the low bandwidth and long delay in the communication channels these systems often has to react autonomously for the environmental stimuluses. Based on the above characteristics of the area, our statements about the identified system class are the following:

- These systems are strongly distributed reactive autonomous systems.
- Embedded, fault tolerant and highly reliable – mission critical – systems.
- These systems often have only very limited processing storing and communication resources.

Due to the above described specific characteristics of the field, in space system development the interface level simulation of onboard equipments has a special importance due to two main nature of the field. The space system developments are done parallel by a number of different – often geographically separated - developer teams. While on the other hand the application of special hardware elements and the lack of maintainability of these systems drives the necessity to ensure the reliability of the system components as well as the whole system.
As the model of an equipment serves as the base of the simulation, one of the key issues is the implementation of the unit model. A well defined and carefully implemented Domain Specific Modelling Language (DSML) provides the opportunity not only to easily build unit models on the necessary level of abstraction but to use formal or semi-formal methods for model verification as well.

GENERALIZED SIMULATION FRAMEWORK

Our concepts of the simulation framework [11] can be described by a 5 layer architecture where each layer has its own well defined responses. From a bottom-up point of view our system is built up by the following layers:

- Physical layer – unit connectors – responsible for interconnecting the onboard units.
- The unit interface simulator layer's main responsibility is to convert the uniform internal communication data structures into the physical layer's communication entities.
- The message broker layer – message queues – responsibility is to serve as buffers for the communication messages and to ensure the chronologically correct message delivery.
- The unit simulation layer performs the simulation of the reactive and autonomous behavioural logic of the onboard equipment based on its domain specific model (DSM).
- The topmost layer is the data presentation and control layer which serves as the main interface between the user and the simulation environment.

In our vision one of the main benefits of the above structure is that: starting from the bottom ones the layers are dynamically interchangeable with the real onboard system elements in different phases of the development.

Unit Behaviour Simulation Layer

The unit simulation layer is the most complex layer in the simulation environment. As its main responsibility is to perform discrete event simulation [6] over the model of onboard equipment [7], this is the only unit dependent layer. This dependency can be reduced by utilizing the benefits of a well-defined and suitable modelling language used to describe the behaviour of the unit on the proper abstraction level. From our point of view the well-definedness of the modelling language means that the semantics of the basic entities of the language are defined correctly. A suitable language should meet the requirements of the space systems area. Our requirements against the modelling language are the following:

- The language should be capable of describing the behaviour of a reactive autonomous system including the logical interface description and the internal computation logic.
- The semantics of the building blocks should be well defined. Only this can ensure that formal and semi-formal methods can be used for running model checking and correct simulation over the unit model.
- As the behavioural description of an autonomous reactive system is often easier by distinguishing different internal states the language should support state machine based description. The behaviour description in a specific state of the unit should support activities, conditional statements, and the definition of periodical-,

sequential- and parallel control flows. Furthermore the activities should be divided into two main classes: internal activities describe the state changes while external activities describe the communication flow of the unit.
- The state machine based description should distinguish between nominal and non-nominal states for support the fault injection and fault propagation investigations.

The general modelling languages like UML are often lack of correct semantic definitions. We decided to implement a so called domain specific modelling language (DSML) [8] which satisfies all of the above requirements and can be easily maintained and used. The fundamentals of this modelling language have already been introduced [4].

THE ELABORATED MODELING LANGUAGE

Inside the earlier described system class the following major problems were identified

- Parallel development of the different units may require the early harmonization of the overall system characteristics to avoid design failures in integration phase.
- Long term missions often operates "out of the reach" and requires autonomous and reactive behaviour.
- Limited resources drives the necessity to inspect the resource usage (power, bandwidth, memory space, etc.) in early phase of the mission planning.
- Robustness and fault tolerance requires checking the behaviour of the entire system in non-nominal states.

From our point of view the lack of well-defined semantics of UML diagrams [9] makes them unsuitable to use as the high level modelling language of the behavioural simulation layer inside the previously described framework. To avoid the ambiguity of the models a well-defined and semantically anchored modelling language – a so called domain specific modelling language (DSML) – should be elaborated and utilized. Designing a modelling language (ML) covers the following tasks:

- defining the concrete syntax
- defining the abstract syntax
- defining the semantic of the ML
- defining well-formedness rules
- defining transformations for other languages

The concrete syntactic of a model describes the appearance of the elements of the language. The abstract syntax defines the elements- the attributes of the elements- and the relationships between the elements of the language. The abstract syntax is generally defined in the form of a metamodel (model of the model), with semantics associated to the metamodel. In the following sections we are going to describe the semantics of the abstract and concrete syntactic and the well-formedness rules of the elaborated modelling language.

The abstract syntax and semantic

The elaborated modelling language uses a two-level hierarchical modelling approach. At higher level the modes- and mode transitions of the onboard equipment can be modelled using UML state machine like diagram. This diagram is called the mode transition diagram and it describes the modes of the modelled unit from a specific aspect. The lower level model is used to describe the behaviour of the activities defined at the higher level model by specifying the action firing sequences of each activity. The lower level model uses UML activity diagram like notation for describing the control sequence of the basic operation.

Mode transition diagram

Using the mode transition diagram the unit model can be described as simple (e.g. hierarchically non refineable and not orthogonal) modes and transitions between modes. In this level of description the mode transitions do not have semantic denotation, their only purpose is to make easier to survey the model. In order to define a formalized modelling language the semantics of the elements of the language should be specified. In the case of the mode transition diagram this means that the internal activity firing semantic and the mode transition semantic should be described using formal specification. This called semantic anchoring [10] which means that the element of the model should be mapped into the semantic domain of the modelling language.

The complex operations (activities) can be defined inside the modes. Unlike the original model the modified state-machine does not allow to describe control sequences at this level. Activities have the following types:

- **Entry**: step into a mode fires the entry tagged activities immediately. As the activities can be delayed by a specific amount of time the firing of an activity does not mean the execution of it rather the scheduling of the activity.
- **Exit**: step out a mode makes the exit type activities to run immediately. An exit type activity always runs before the new state entry activities.
- **Periodic**: runs periodically inside a mode. After the period time expires and a periodic activity terminates the activity become fireable again.
- **Event**: an event triggered activity.

The activities have more attributes depending on the type (for example period time and delay for periodic activities or just delay for the event activities). These can also be defined on the level of the mode transition diagram.

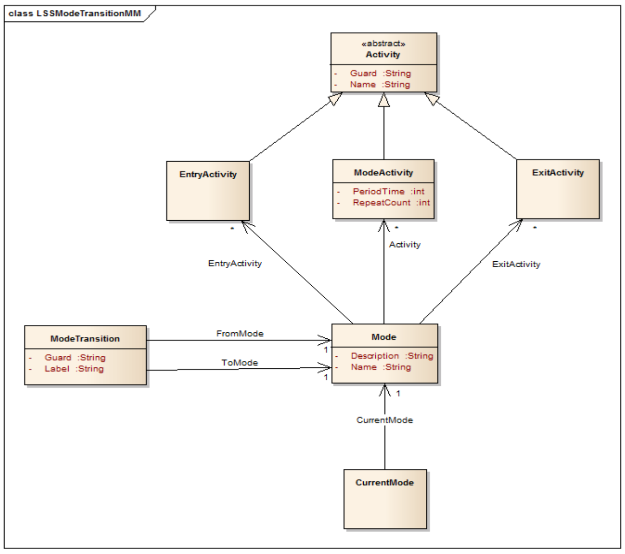

Fig. 2. The metamodel of the mode transition chart

Figure2. shows the metamodel (abstract syntactic) of the higher level model. Our mode transition model has 4 different elements, the modes, the mode transitions which have references for the source and destination state. One state has arbitrary number of activities.

Action flow diagram

At the lower modelling level – action flow diagram level – the control sequences presented in the original model can be described. The elements and relationships between them are the following as shown on Figure 3.

- **Action:** Actions are the same as in the original model. They describe basic operations of the simulation. Actions can refer to reading-writing values on bus, to manipulate control signals and actions can trigger mode changes as well.
- **ControlFlow:** Control flow describes connection between actions on the model. It defines the control sequence. The control flow describes the flow of execution token inside the model. An action which receives execution token from its predecessor fires immediately and passes the token to the successor actions.
- **Condition:** A condition element holds decisions for controlling the behaviour of the model depending on either external or internal conditions. Loops can also be defined using condition elements.

The basic constraint made for the metamodel is that if the control flow graph of the diagram contains a fork node than the graph should not contain a road, from its initial point to its termination point, which does not contain the same fork node. A joint node has always to be between two fork nodes. We used object constraint language (OCL) to describe the above constraint.

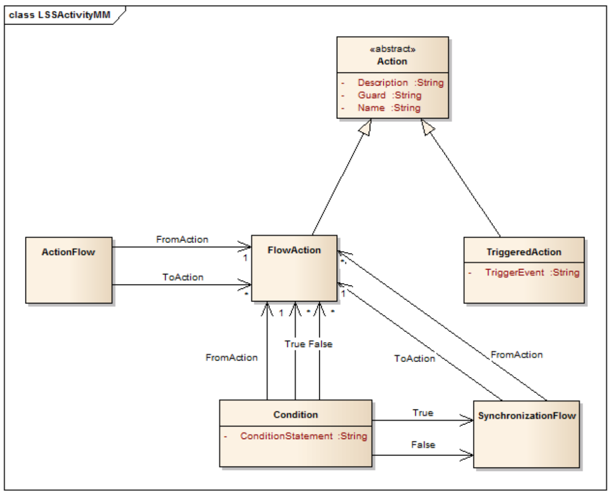

Fig. 3. Metamodel of the activity chart

The concrete syntactic

The design of metamodel is followed by the design of the concrete syntactic of the modelling language. The concrete syntactic defines the appearance of the elements introduced in the metamodels above. Example of the concrete syntax of the DSML is presented on figure 5 and 6. The EMF generates automatically java classes for the manipulation of the model from the implemented metamodel. Using the metamodel and the generated classes we implemented a graphical editor for our ML under graphical modelling framework (GMF).

Simulator architecture

The behaviour simulation layer is responsible to run the simulation of the unit models. The behaviour simulation layer is capable to run parallel simulation of multiple unit models and provides a built in mechanism for the simulation of inter unit communication. The behaviour simulation layer is capable of simulating multiple units of a distributed on-board system.

The unit behaviour simulator provides a global workspace shared by every simulated units and unit specific workspaces for each unit. A unit model may contain multiple mode transition models which are executed by the simulator in parallel inside the same unit specific workspace. Figure 4. shows the internal architecture of the behaviour simulator. The benefit of the architecture is that the same simulation environment is capable of simulating either only 1 unit or more units of the on-board system based on the actual simulation aspects.

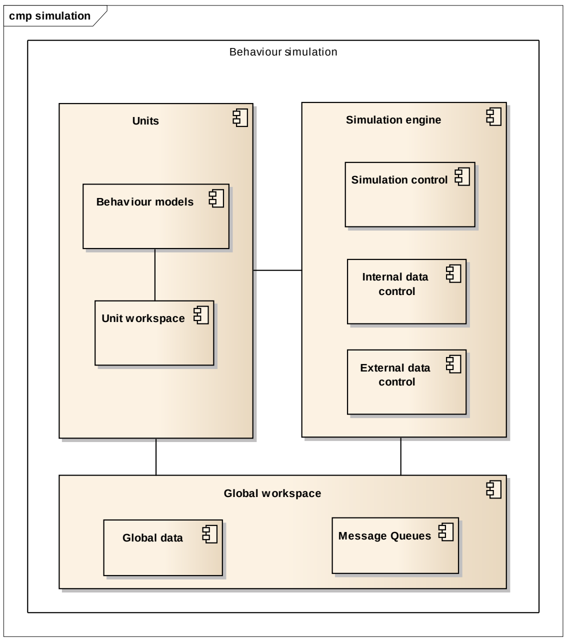

Fig. 4. Internal architecture of the behaviour simulation layer

A CASE STUDY

For a case study we have chosen to implement the model of an on-board camera system. The camera system contains an imaging detector and a rotation mechanism which can rotate the detector. The detector and rotation mechanism is controlled by the internal logic of the on board unit. The control logic of the camera system implements the following major features:

- Rotate the detector into predefined positions
- Take an image – read out the detector data
- Process image before sending it to the space craft on board data handling system
- Generate and send periodic housekeeping data to the space craft on board data handling system
- Receive and process incoming commands from the space craft on board data handling system

The goal of the simulation was to evaluate the behaviour of the instrument control logic from the following aspects:

- Power consumption
- Memory consumption
- Science data load estimation

Behaviour Model of the Control Logic

Our goal during the simulation was to simulate single on-board equipment (unit) to evaluate the memory consumption during operation. The aim of the simulation is to estimate the optimal balance of the memory requirement and image preparation frequency. The unit model contains two parallel mode transition diagrams and the corresponding action flow diagrams. Using multiple mode transition diagrams the parallel and cooperative execution can be described in a simple and convenient way. The multiple mode transition diagrams are executed in the same unit specific workspace by the simulation engine. Figure 5. shows the internal – detector and actuator control – mode transition diagram of the on board equipment while figure 6. shows the external – communication – mode transition diagram.

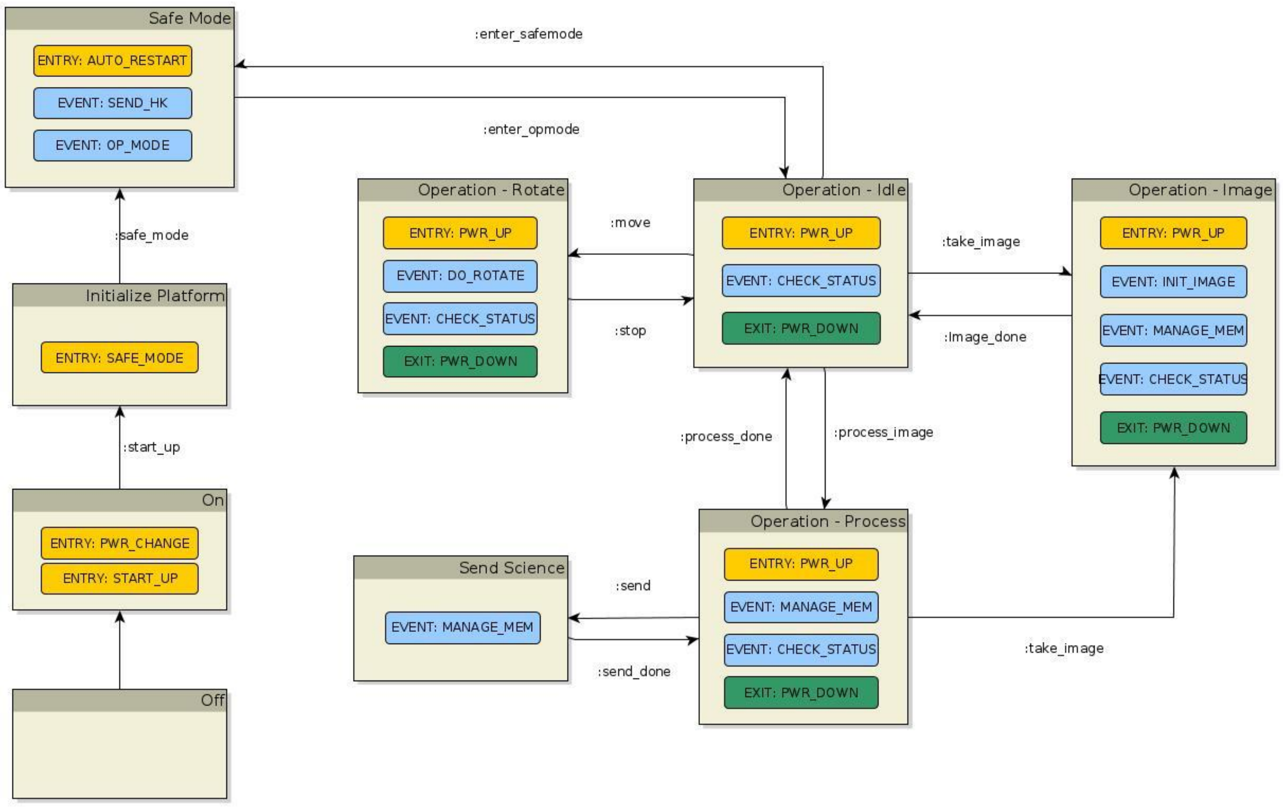

Fig. 5. Mode transition diagram of the on board unit

The instrument control logic has a bootup sequence, a safe mode and an operational mode with take image, process image and rotate submodes. Parallel with these modes the instrument continuously accepts commands and generates housekeeping data. The semantics of the modelling language guarantees that mode transitions guarded by the same expression inside one workspace are fired in parallel. The simulation engine ensures that the exit action flows of the multiple source modes are executed prior to the entry action flows of the target modes.

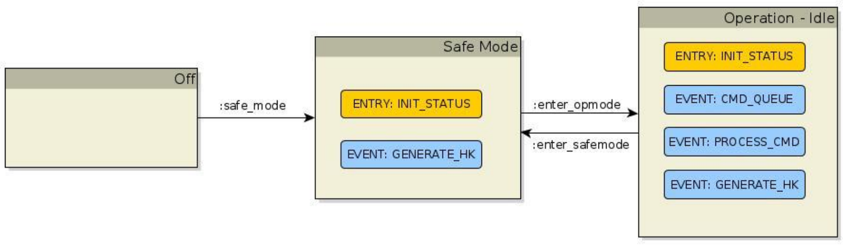

Fig. 6. Mode transition diagram for describing the communication of the on-board unit

CONCLUSION AND FURTHER WORK

So far in this paper we discussed a semantically anchored domain specific modelling language elaborated especially to use in the field of autonomous distributed on board system simulation. The elaborated modelling language is capable of describing the interface level behaviour of on board equipments from different aspects. We presented a power and memory consumption and scientific data load model of an onboard equipment of the Mars TGO. Using formalized models of onboard equipment description is a corner stone in a successful development process as it is suitable for:

- Define basic specifications for the unit.
- Identify design failures in the early stage of the development – long before the integration phase.
- Use the formalized model to verify the behaviour of the implemented unit.
- Use the formalized model and simulation environment for automatic test generation and execution.
- Examine the system behaviour in non-nominal states.

REFERENCES

[1] G. Tróznai, A. Baksa, S. Szalai, B. Sódor: Rosetta Lander Software Simulator, 57th, International Austronautical Congress Valencia, Spain – 2-6 October 2006, IAC-06-D1.P1.11

[2] G. Tróznai, A. Baksa, S. Szalai, B. Sódor: Spacecraft Simulator for Philae, 9th, International Workshop on Simulation for European Space Programmes, Noordwijk, The Netherlands – 6-8 November 2006

[3] B. Sódor, G. Tróznai, Cs. Lipusz: Implementing Data Presentation Layer In Testing And Simulation Environments using XML 58th, International Austronautical Congress Hyderabad, India – 24-28 September 2007, IAC-07-D1.I.11

[4] B. Sódor, A. Baksa, A. Balázs, S. Szalai, G. Tróznai: New Approach to Modelling Spacecraft Modules. 58th, International Astronautical Congress Glasgow, Scotland 29-september – 3-october 2008

[6] Fishman, G. S.: Principles of Discrete Event Simulation. Wiley, New York, 1978.

[7] William Delaney, Erminia Vaccari: Dynamic Models and Discrete Event Simulation. Dekker INC. 1988.

[8] Kelly, S. and Tolvanen, J-P.: Domain-Specific Modeling: Enabling full code generation. John Wiley & Sons, ISBN 978-0-0470-03666, 2008, 427 p.

[9] G. Pintér: Model Based Program Synthesis and Runtime Error Detection for Dependable Embedded Systems, PhD. thesis, BME, Department of Measurement and Information Systems, 2007

[10] Kai Chen, Janos Sztipanovits, Sandeep Neema: Toward a semantic anchoring infrastructure for domain-specific modeling languages, Proceedings of the 5th ACM international conference on Embedded software, September 18-22, 2005

[11] Balint Sodor, Gabor Troznai, Sandor Szalai, Dr.: A DSML based Approach for simulating on-board Equipments in Space Applications. Proceedings of the 2012 SESP workshop.