Open Archive TOULOUSE Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 12362

To cite this version : Dragomir, Iulia and Ober, Iulian and Lesens, David *A Case Study in Formal System Engineering with SysML*. (2013) In: IEEE International Conference on Engineering Complex Computer Systems - ICECCS 2012, 18 July 2012 - 20 July 2012 (Paris, France).

Any correspondance concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

A case study in formal system engineering with SysML

Iulia Dragomir, Iulian Ober
University of Toulouse / IRIT
118 route de Narbonne, 31062 Toulouse, France
{*iulia.dragomir, iulian.ober*}*@irit.fr*

David Lesens
EADS Astrium Space Transportation
66, route de Verneuil - BP 3002, 78133 Les Mureaux, France
david.lesens@astrium.eads.net

***Abstract*—In the development of complex critical systems, an important source of errors is the misinterpretation of system requirements allocated to the software, due to inadequate communication between system engineering teams and software teams. In response, organizations that develop such systems are searching for solutions allowing formal system engineering and system to software bridging, based on standard languages like SysML. As part of this effort, we have defined a formal profile for SysML (OMEGA SysML) and we have built a simulation and verification toolbox for this profile (IFx). This paper reports on the experience of modelling and validating an industry-grade system, the Solar Generation System (SGS) of the Automated Transfer Vehicle (ATV) built by Astrium, using IFx-OMEGA. The experience reveals what can currently be expected from such an approach and what are the weak points that should be addressed by future research and development.**

***Keywords*-system engineering, modelling, SysML, simulation, model-checking, temporal properties, abstraction**

I. Introduction

One of the major risks of errors in the development of real time critical embedded software is the misinterpretation of the system requirements allocated to the software. These misunderstandings between the system team and the software team may have several sources, but are very often due to the following causes:

- Use of ambiguous means to describe the system requirements and the software implementation, leading to different interpretations by the system designers, the software developers and the reviewers.
- Insufficient knowledge by the software team of the formalisms and jargon used by the system team, leading to the development of software that does not satisfy the system requirements.
- Insufficient knowledge by the system team of the formalisms and jargon used by the software team, leading to inefficient reviews of the software specification and code by the system team.

The errors potentially introduced during the development are then generally discovered very late by very costly test processes.

This situation is a cause of particular concern in the aerospace domain, where multiple suppliers have to build integrated systems based on common system requirements. It has lead the main actors of the field (agencies and main contractors) to search for solutions allowing formal system engineering and system to software bridging. One such effort was the ASSERT project[1] led by the European Space Agency (ESA), with the objective of defining a complete system/software engineering process for complex and critical space systems. The ASSERT development process is now supported by a set of tools called TASTE [9]. The main scope of the ASSERT project was the management of non functional requirements, from the capture of system requirements to the automatic generation of code. The present work is part of a follow-up project, Full MDE[2] [8], which aims to complement the ASSERT process on the functional aspects, allowing full functional system modelling based on an adapted use of the SysML standard [21].

SysML is a graphical modelling language well adapted for system engineering and it is increasingly used in industry in the development of real-time embedded systems. For example, Astrium Space Transportation has deployed SysML for capturing the systems requirements of the new version of the Ariane-5 launcher. But its semantics is underspecified in order to preserve the generality of the language. The OMEGA Profile [13], previously defined for UML [23], has been adapted to SysML in order to suppress any ambiguities and provide a formal operational semantics, rendering SysML models amenable to formal verification. The UML version of the OMEGA profile was conceived to be used in the design and validation of real-time embedded systems, and it has been used in several industry-grade models (e.g., [18], [20]) providing interesting validation results. Like the UML profile, OMEGA SysML is implemented in the IFx toolkit[3] [19] which provides simulation and timed-automata based model-checking [7].

In this paper we assess the use of OMEGA SysML and IFx in the design and validation of the system models of a space vehicle control subsystem: the Solar Generation System (SGS), which manages the deployment and positioning of the solar wings of the Automated Transfer Vehicle (ATV)[4] developed by Astrium Space Transportation for the

[1] ASSERT was a European FP6 Integrated Project lasting from September 2004 to December 2007. http://www.assert-project.net/

[2] Full Model Driven Development for On-Board Software, Contract ESTEC 22618/09/NL/JK

[3] http://www.irit.fr/ifx

[4] http://www.esa.int/atv

European Space Agency. The case study, led by Astrium, covers the entire system engineering phase of SGS; however, in this paper we concentrate on the aspects that are specific to OMEGA SysML and in which the IFx toolbox plays a role. The OMEGA SysML profile was used in reverse engineering to capture the system architecture, covering software, hardware and external subsystems interacting with the SGS, and to give a high-level functional model for each of these pieces. In terms of SysML, the architecture description takes the form of block diagrams, and the functional description of the blocks takes the form of detailed state diagrams, including timing and lower-level action details. The overall system requirements are also formalized so that their satisfaction can be determined by simulation and model-checking with the IFx toolbox.

The case study shows how a formal system engineering approach helps increasing the confidence in system design models, which are used as starting point in the subsequent development steps. The purpose of this paper is to reveal what can currently be expected from such an approach and what are the weak points that should be addressed by future research and tool development.

Paper structure: In Section II we position the IFx-OMEGA approach in the Full MDE Process, while Section III presents an overview of the OMEGA SysML Profile. Section IV provides a brief description of the architecture and main functionalities of the SGS case study. Section V presents the verification model and the properties this model has to satisfy. The validation methodology and the main results obtained by simulation and model-checking are described in Section VI. In Section VII we position our approach with respect to other toolsets, before concluding.

II. The Full MDE Process

The Full MDE project aimed at filling the gaps between system modelling and software modelling, and between the code generated automatically from the software model and manual code corresponding to functional requirements, by using a Model Driven Engineering (MDE) approach.

In the Full MDE process, each step of development of embedded software is formalized by safe and non-ambiguous languages which also provide proof techniques. The Full MDE approach has selected:

- for system design, the OMEGA SysML Profile which is also used in the ASSERT process to capture non-functional requirements;
- for software design, SCADE suite [34] - a graphical modelling language mixing automata and data flow views - and
- for implementation code, the SPARK Ada language [35] designed to support the development of software systems where the correct operation of software is essential for the security and safety of the system.

The successive refinement steps between the three model layers are realized by model transformation techniques.

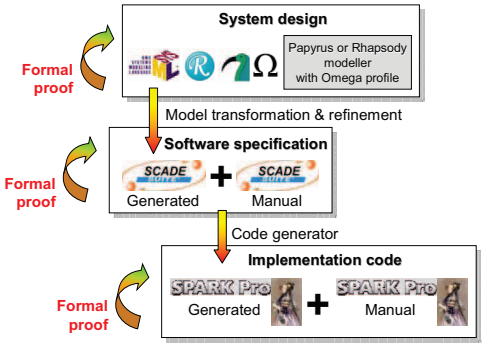

Figure 1. FullMDE: An engineering process based on models, formal methods and model transformation

The purpose of a formal approach is to verify as early as possible by simulation and proofs the system and to discover potential errors, an approach that is more reliable and potentially less costly compared to a more traditional one relying only on informal paper documents and on tests.

Figure 1 summarizes the Full MDE process.

The system design is an important task for the development of critical systems and it guides the entire process. The architecture of the system and its properties are expressed in the OMEGA SysML Profile and are subject to formal validation within the IFx toolbox. Once it is verified, the model is transformed and integrated into SCADE modeller which verifies the complementary sets of requirements. SPARK Ada code is then generated automatically from SCADE by a certifiable tool. The combination of SPARK and SCADE allows the verification of correct integration of manual and generated code, the absence of runtime errors and the satisfaction of functional properties. A more detailed description of the process can be found in [8].

III. Overview of OMEGA SysML and IFx

OMEGA SysML defines the semantics of a rich subset of SysML elements, providing the necessary constructs for modelling real-time embedded systems designs. The profile also defines a very few extensions, in the form of *stereotypes*; their use however is not mandatory and a SysML model can generally be interpreted as an OMEGA SysML model without syntactic changes.

The architecture of a system is defined in terms of blocks; OMEGA supports most of the standard features of SysML blocks, such as properties (attributes and parts), relationships (associations, compositions and generalizations), interconnection elements (connectors, ports, interfaces) and mechanisms for defining structured data types and signals. As usual in SysML, such elements are described in Block Definition Diagrams (BDDs) and Internal Block Diagrams (IBDs). Being a formal profile, OMEGA defines a set of additional well-formedness rules, in particular for strong static typing. The profile also clarifies the semantics of

elements that are not fully described in the SysML standard, such as the default behaviour of ports. The well-formedeness rule set is fully formalized in OCL [24] and is enforced by the IFx tools, which encourages rigorous system engineering. Further details on the rules, their rationale and their formalisation can be found in [17].

The behaviour of system components is expressed in OMEGA by *state machines*, possibly involving detailed *action* descriptions, which can be structured into *operations*. Actions are expressed in concrete textual semantics, compliant with the UML action metamodel, and cover notions like: variable assignments, signal output, block and connector creation/destruction, operation calls, expression evaluation, value return. As the standard action language of the OMG, ALF [25], is under work, there are plans to use ALF as action syntax in OMEGA in the future.

OMEGA defines a timed operational semantics, based on the asynchronous composition of the behaviours of different blocks. In order to allow the user to control the concurrency granularity, which can have an important impact on the effectiveness of the verification tools, the semantics partitions the set of blocks into *activity groups*, based on the concurrency meta-attribute (a standard feature of SysML). Thus each block can be defined as active or passive, and each active block defines an activity group consisting of itself and its dependent passive blocks. Activity groups are executed concurrently and they react to external stimuli (signals) in atomic run-to-completion steps. Details on the execution and communication model can be found in [16].

The model time base can be either discrete or dense and it is specified by the user when simulation or model-checking is performed. Model time is controlled using the primitives of timed automata with urgency [5]: clocks, timed guards and transition urgency annotations, which are stereotypes used to control how logical time can progress depending on the transitions that are fireable in a certain state. Time progress can thus be disabled (when at least one *urgent* transition is fireable), enabled but bounded (when there is no urgent transition but there is at least one *delayable* transition), or enabled and unbounded.

One of the few syntactic extensions defined in the profile concerns *observers*, which are used to formalize the system requirements that correspond to timed safety properties. An observer is a special type of block (stereotyped with ≪observer≫) that can monitor the system states and events and give verdicts about the (non-)satisfaction of a requirement. Its behaviour is defined by a state machine, with some states qualified as ≪error≫, respectively ≪success≫, states expressing the (non-)satisfaction of a safety requirement.

The SysML Profile is implemented by the IFx-OMEGA toolbox, which provides static checking, compiling, simulation and model-checking tools. The tools rely on the translation of the relevant parts of a model to the input language of the IFx Toolset[5] [6], which is based on asynchronous communicating timed automata.

IV. The ATV Solar Generation Wing Management System

The ATV, developed by Astrium Space Transportation for the European Space Agency, is a space cargo ship launched into orbit by the European heavy launcher Ariane-5 with the aim of resupplying the International Space Station. The case study presented here - the Solar Generation Subsystem (SGS) - manages the solar arrays which provide the vehicle, for up to 6 months, the energy needed to fulfil its mission. SGS is responsible with providing functional chains to realise the solar arrays deployment and controlling the rotation of the solar wings of the ATV.

The SGS consists of a set of hardware and software entities; its software part is a subsystem of the Flight Application Software. The hardware of this subsystem has more than 70 devices, with multiple levels of redundancy for achieving reliability and availability in case of failures. Its main elements are the four wings, each of them being powered by two motors; a set of four motors (one from each wing) is controlled by a solar array driving electronics (SADE). Each wing has four hold-down and release systems (HDRS) which maintain the wing in its initial stowed position during the Ariane-5 flight. To deploy a wing, each HDRS has to be cut by a thermal knife; there are two knives for each HDRS, a nominal one and a redundant one. All these hardware pieces are enabled at the right time by four power units and four thermal control units. The activation of each piece of hardware is given by two command and monitoring units which also supervise the entire system. Each thermal control unit has 2 regulators - solar arrays driving group (SADG) -, which control 4 thermal knives each. Each regulator cannot power more than one thermal knife at a time.

The system has two operating modes:

1) The solar wing deployment mode. During take-off the four wings of the ATV are stowed. Their deployment starts by removing the safety barriers from the thermal control units (TCU). Safety barriers prevent the unwanted release of a wing by forbidding the enabling of the thermal knives by the TCUs. Then the restraint cables are cut by at least 4 of the 8 thermal knives of the wing. In order for the HDRS to be cut, each knife has to be active for 50 seconds. The deployment of the panels starts immediately after the last hold-down system has been cut. The deployment motion is driven by deployment strings at the root hinge and the panel hinges. After the deployment is completed, the panel hinges and root hinge lock positively to provide a stiff, backlash free, deployed configuration and safety barriers are restored.

[5]http://www-if.imag.fr/

2) The solar wing rotation mode. It is realized by two motors that control the position of the wing and connect it with the avionics equipment chain. The position of the wing is measured by reed switches and magnets fixed on the wing's motors. The solar array driving electronics that control the wings are working in cold redundancy: at a given time only one can be on and control the orientation of the 4 wings. The measures captured by the sensors are sent via the active SADE device to a central processing unit which numerically computes the direction in which the wings should be rotating and their speed, results sent back to SADE to be executed. The rotating mode is infinite, by cyclic activation of one SADE device.

V. The system model

In the scope of the Full MDE R&T project, the SysML model has been built in reverse engineering by the Astrium engineering team using IBM Rhapsody[6]. In this section we describe the architecture of the model, the structural issues that have been detected by applying the OMEGA well-formedness rules and the changes they induced in the model, and the formalization of the dynamic property that was verified.

A. The original system model

The SysML model captures both the architecture of the system and its timed behaviour. There are three types of blocks:

- 9 *hardware* block types describe the equipment of the ATV. Their behaviour is generally split in two operating modes: a *healthy* mode which models the nominal behaviour and a *failure* mode.
- 3 *software* block types, each with a specific function (e.g. deployment, rotation, etc). Each block reacts to stimuli coming from the environment (external mission management) and controls the behaviour of the hardware devices by executing automated procedures.
- A block that "simulates" the mission management. It initiates the two functionalities of the ATV solar generation system (deployment and rotation of wings) respecting the timing constraints imposed by each of these modes. In order to validate the software, the mission management models a finite behaviour - the wing deployment and three wings rotations.

Some blocks are further decomposed in sub-blocks, the maximum depth consisting of 4 layers. The model defines a total of 20 block types (7 of which are refined by means of 24 Internal Block Diagrams), 204 ports and 234 connectors. The entire system contains 95 block instances: 12 composite ones refined into 78 hardware devices, 3 software instances, one instance of the mission management and the one representing the entire system; with a total of 517 port instances and 366 connectors.

[6]http://www-01.ibm.com/software/awdtools/rhapsody/

B. Static coherence and compliance with OMEGA

The first benefit of applying the Full MDE process to this SysML model comes from the application of the static well-formedness rules of the OMEGA profile, which allows detecting a range of inconsistencies, in particular related to typing. Indeed, standard SysML modelling tools like Rhapsody do not impose strong static typing rules, and accept for example the definition of untyped ports. Since ports are not typed, connectors can also be defined between potentially incompatible ports. Such errors can go unnoticed throughout the system engineering phase, and last sometimes until the integration test campaign, when correcting them becomes very costly.

The initial SGS model did not contain any interface definition, and all the 204 ports were untyped. The errors provided by the OMEGA compiler have led to the definition of 18 interfaces which were used to specify port contracts. This allowed us also to detect connection errors in the model, such as duplicated connectors. We found that around 20% of the ports defined throughout the model were not respecting the uniqueness and completeness rules, which means that the destination of an incoming signal was either undefined or ambiguous. This led to another reiteration on the system model either correcting the contract for these ports or splitting their functionality with respect to the target object.

As a counterpart, several syntactic changes were required for static compliance to the OMEGA profile. For example, OMEGA requires unique names within a block or state machine: an attribute and signal reception having the same name or two sub-states within two parallel states of the same statechart with the same identifier are forbidden. The syntax for certain constructs (timers, enumeration literals, access to arrays, etc.) is different from the one used in Rhapsody/C++. However, the effort required for these syntactic changes is minor relative to the size of the project: between 1 and 2 person*days for the SGS model.

C. Preparing the model for formal verification

Once the model is statically coherent and compliant with OMEGA, some effort is still necessary to prepare it for formal verification, in particular for formally defining the properties to be verified. The main objective, for SGS, was to verify that the system is 1-fault tolerant: the mission succeeds even if one component of the system has failed during execution.

Injecting failures into the system model. The failure mode was already modelled for each of the 62 hardware devices of the SGS: thermal knives, hold-down and release systems, thermal control units, solar arrays driving groups, power units and command and monitoring units. However, in order to ease the definition of verification configurations, we decided to add a block that commands the failure of one equipment, chosen based on a parameter that can be configured before each verification session without recompiling

the whole model. We have added another 144 ports and 138 connectors for inducing errors. The target device fails non-deterministically, at an arbitrary moment during the system execution.

Formalizing system properties. Requirements are captured by OMEGA observers, then verified against the model under the 1-fault hypothesis.

The requirement we describe here is defined with respect to temporal properties of the system and is concerned with the behaviour of the system in case of failure: no matter what error for equipment devices is injected in the system, the deployment of the wings shall succeed. This implies that the redundant devices replace nominal ones in case of failure.

Property 1: After 10 minutes since SGS start-up, all four wings are deployed and the mission management is aware of it.

The observer in Figure 2 models this property. In the initial state the system is off. Because we need to observe specific objects from the system and these objects exist only after the initialization of the entire system, the observer waits until the *initialized* signal is emitted by the *ATV*. Then it triggers the transition and arms a clock with the value *deployment_duration*, in this case 10 minutes. If after this deadline the four wings have not been deployed, the observer enters the state *NOT_DEPLOYED* and the property is violated. If the wings have been deployed, the observer enters the state *DEPLOYED* in which it verifies if the mission management is aware of it. After the entire deployment process, the software piece concerned by this task verifies the locked status of the wings, via an intermediary control and monitoring unit (CMU), and forwards the global confirmation to mission management (MVM). The MVM resumes its behaviour and continues its mission until the *END* state is reached. Then the observer enters the success state *MISSION_EVENT* and the expressed property is satisfied. If it is not the case in the time passed from start-up then an error has occurred and the property is violated.

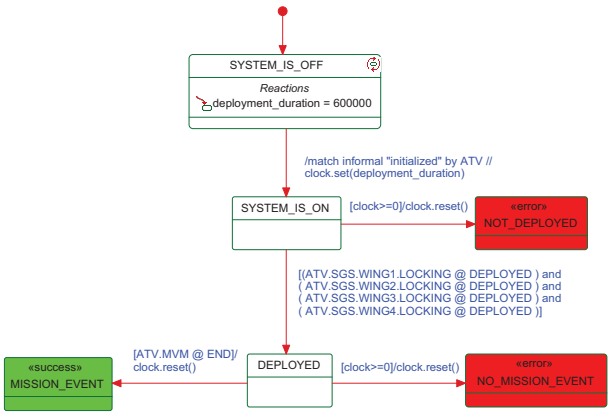

Figure 2. Observer modelling that the four wings are deployed 10 minutes after start-up and the mission management is aware

VI. System validation

A. Validation methodology

In IFx-OMEGA, the validation of models consists in performing several steps, supported by different tools, according to the following workflow:

1) The first step is the translation phase from the OMEGA model to an IF model by a *sysml2if* compiler, which enforces the well-formedness rules defined by the profile and allows detecting some static errors, as already discussed in Section V-B.
2) One can proceed with an interactive random simulation phase, which allows us to quickly detect certain problems, such as deadlocks that occur systematically.
3) The correct modelling of the system can be verified via the absence of deadlocks. It is an extension of the simulation phase in which we can pre-check the system model by searching for certain state space configurations.
4) The main validation step is the model-checking phase which, in IFx-OMEGA, consists in the exhaustive exploration of the product state space of the relevant part of the model and a set of observers while searching for the absence of *error* states and stopping at *success* state. If the model-checking algorithm finds errors then it produces scenarios which can be replayed in the simulator, in order to diagnose the cause.
   The verification is usually not a one-shot phase, different strategies for reducing the combinatorial explosion of the state space being often applied in an iterative fashion. In IFx-OMEGA, some reduction of the state space can be obtained automatically, for example by applying conservative model transformations based on static analysis techniques (dead code/variable elimination, variable abstractions) or by applying partial order reduction [12]. However, it is often necessary to involve the user for manually defining abstractions, as we will show in the case of SGS.

B. Validation by simulation

Despite the fact that the SysML model had been thoroughly reviewed, the interactive simulation of nominal scenarios and the execution of a series of random simulations allowed several modelling errors to be found.

Most errors discovered in this phase concern unexpected message receptions. When a block receives a message while in a particular state when the message is not expected, it will block and this generally leads to a global deadlock or model failure. In the SGS model, several such problems have been found in the behaviour of thermal knives and of wings. Some unexpected message receptions had to be modelled.

In some cases, like for the SGS mission management block, the unexpected message receptions were due to the

existence of parallel composite states reacting to the same message; the designer intended only one parallel state to react to the message during a certain flight phase, but failed to specify correctly the conditions that enabled and disabled the receptions when going from one phase to another. Even if the model is intended to be a high-level system description, such behaviours can be quite intricate and errors are difficult to find without a simulation tool.

We mention that the length of a SGS simulation scenario is around 2400 steps (a step corresponding to the firing of a transition), and needs less than a minute to be executed on a regular desktop computer.

C. Verifying the absence of deadlocks

Even though a large set of deadlocks was detected and corrected during the simulation phase, in unexplored states of the system other deadlocks might appear. These ones concern the hardware devices which can block if the software parts send inadequate (sequences of) requests. The absence of deadlocks ensures only that the system is correctly modelled and does not guarantee that the system correctly behaves. Finding the modelling errors that result in deadlocks is an important task since we can detect unrealistic behaviours of the modelled systems that often go unnoticed due to incorrect interpretations of the model until software integration.

It was the case for solar array driving electronics (SADE), responsible for the wing rotation, which failed due to an incorrect deactivation sequence: the deactivation command preceded the disabling command. The two signals were sent in this order by the software: deactivation via a power unit and disable directly, without any timing constraints being imposed. This error was due to an unrealistic approximation made in the model, namely that the deactivation command is executed in zero time. In the real software, the deactivation command has a 200 ms execution time thus leading to a correct switch-off of the device. The correction consisted in adding a timer for the deactivation command in the power unit.

No other deadlocks were found after this verification step, under the hypothesis of no induced failures.

D. A failed model checking attempt with positive results

The next validation step was to attempt to verify, by model-checking, the property described in Section V-C. This requirement had to be verified against the 62 possible failures of hardware devices. In the current configuration of the system, the verification of this requirement does not finish[7].

We chose to continue by using a non-exhaustive exploration strategy: the system is executed on only two threads, one thread for the simulation block which induces failures and the second thread for all the other SGS blocks. This execution is partial in the sense that signals sent to different blocks are always handled in the order in which they were sent and different action interleavings are not taken into consideration. But, as long as errors are detected, it is an acceptable search strategy, and it allowed finding the three errors detailed below:

- In case of a power unit failure, the violation of Property 1 led us to discover a major error in the model concerning the connections between the power units and the wings. With the initial connections, in case of power unit failure, a wing could not be deployed, because both the nominal and the redundant connection were established between the same power unit and wing. In order to really be one fault tolerant, each power unit shall power two wings, and each wing shall be powered by two units.
- In the case of a thermal knife failure, in a particular configuration, the state machine of the corresponding hold-down and release system could deadlock because two knife OFF signals were received (one from the nominal, one from the redundant knife) but only one was handled.
- In the case of a control and monitoring unit failure, Property 1 is rejected. At the end of the deployment mode, wings' status verification does not finish in the given time and the MVM does not receive the confirmation to continue its mission. This type of error is actually incorrect, since, in reality, the CMU is composed by two independent devices which make it 1-fault tolerant, and it has been removed from the set of possible induced failures. A similar case was found for thermal control unit failures, which have also been eliminated.

These errors are very similar to those found by interactive simulation, but the particular configurations in which they occur makes them very hard to detect without an automatic search of the state space.

E. Verification using abstractions

After the errors described above were corrected, the state space search with concurrency reduced to two threads produced no further errors. However, since the reduction eliminates potentially problematic executions, this is not sufficient to establish the satisfaction of the required property in the general case. To proceed with verification, we set out to use abstractions.

Figure 3 shows the communication graph of the SGS system, generated automatically from the state space explored as discussed before. Nodes represent block instances and arrows represent the existence of at least one signal communication between the block instances at the ends. In order to make it readable, we structure the graph accordingly to the three categories of the system - mission management, software and hardware - and we group the instances of each

[7]We were using a IA64 computing server with 16GB of memory. The state space generation had to be stopped after 3 hours as the memory was exhausted.

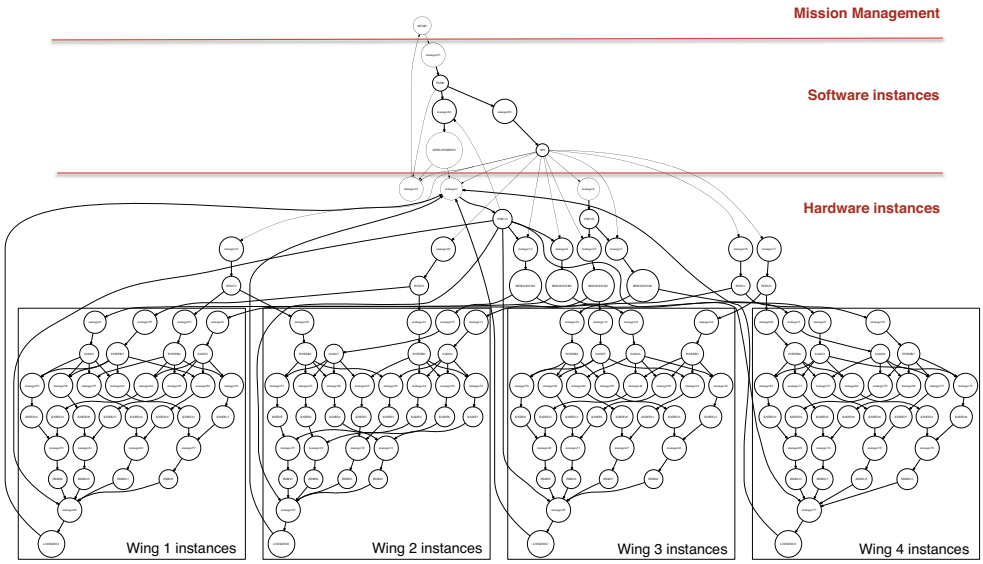

Figure 3. System's communication graph displaying the block instances - represented as nodes - and communications between them - represented as arrows

wing as they represent the key concept of our abstraction. As one can see, each of the 4 wings generate a large structure of block instances, corresponding to the hardware equipment of the wing. However, from the point of view of the software blocks (which are the only other blocks with which it communicates), a wing *that does not experience any hardware fault* exhibits a very simple behaviour: it ends up by being deployed. Therefore, one can set out to verify the property of deployment under 1-fault hypothesis (Property 1) by replacing all the 3 wings in which no fault occurs by an abstract version that models this simple behaviour. This requires to run a separate verification for each one of the 62 faults, using one of four possible versions of the SGS model (one version for every combination of 3 wings that are replaced by an abstract model).

Due to the interconnection of wings and solar array driving groups - members of thermal control units (TCUs), it is actually possible to further group an abstract wing and its corresponding TCU, if the concrete wing is not connected to the same TCU. The corresponding abstraction is represented by a *TCU_WING* block. Thus, after abstraction, each of the 4 versions of the model contains one concrete wing and its TCU, one abstract wing connected to the same TCU, an *ABSTRACT_TCU* block connected to both concrete wing and abstract wing, and two abstract *TCU_WING* blocks. The communication graph of an abstract model is shown in Figure 4. The system is reduced, in average, by 55% with respect to the number of block instances, port instances and connectors.

The behaviour of an abstract wing consumes all received signals and answers *not deployed* for the first 400 seconds since system start-up (the minimal delay needed for a correct deployment) and as *deployed* after 350 seconds since start-up (the answer between 350 and 400 seconds is non-deterministic). The difference of 50 seconds is related to the end steps of the deployment procedures, in which the wing can be deployed or not. The abstracted *TCU_WING* has a similar behaviour, with the difference that needs at least 280 seconds for deployment and can be not deployed for at most 330 seconds. These intervals were manually computed by adding the time needed for each step and trying to minimize their intersection were the wing has a non deterministic behaviour with respect to its status.

The reduced model was model-checked by injecting failures one at a time. Concerning thermal knives, due to the parallelization, we found an error scenario. Deployment starts by switching on a subset of nominal knives for each wing. The knife will inform the hold-down and release system of its enabling. If, in these 50 seconds needed for the cut, the redundant knife (which is off) fails, it will send the disabling command to the hold-down system, and the HDRS won't be cut. Now, the redundant knife cannot be enabled (due to the failure mode), the cable is not cut and the wing not deployed. The correction is to send the switch off request in case of failure only if the knife is on.

For all the other injected failures, the property is satisfied. Table I presents the average time needed for the verification (in seconds) of each class of errors. In the case of the later one, the verification of the induced failure on the power unit responsible for the nominal connection with the concrete wing does not finish, while with the induced failure on the power unit responsible for the redundant connection the

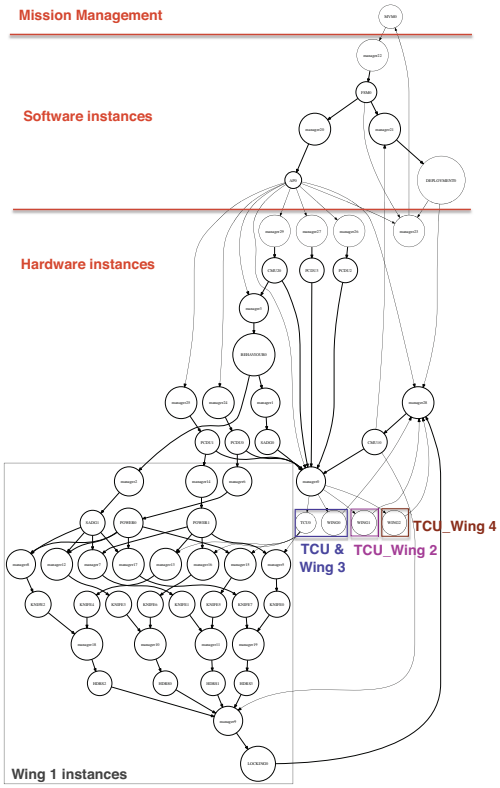

Figure 4. Abstract system's communication graph - the instances for three of the four wings are replaced with simpler components

Table I
AVERAGE TIME VERIFICATION FOR INDUCED FAILURES

Induced Failure	Average Verification Time (s)
Thermal Knife	13993
Hold-down and release system	12272
Solar array driving group	11377
Power unit	∞

requirement is verified in 11394 seconds. This shows the asymmetry of the system model.

F. Towards contract-based reasoning

In order for the verification described in the previous section to be complete, we still need to prove that the abstractions of wings and TCUs that we used are correct. However, this is not possible without making certain *assumptions* about the environment in which a wing operates, which is composed of the mission management, the software parts, the control and monitoring units and the power units. The assumptions concern the order and the timing in which messages are sent by the environment; in their absence, the concrete model of a wing can deadlock or malfunction, which is not captured by the abstraction.

One then has to verify that the environment actually *guarantees* these assumptions, which is true only under the assumption that the wings and TCUs operate correctly as specified in the abstraction. This kind of circular argument is used and can be proved to be sound in certain contract-based verification frameworks [30], [29].

Defining such a contract framework for OMEGA SysML and supporting it in the toolset is part of our future work roadmap. In initial experiments, we have manually represented guarantees as observers and verified each guarantee for the concerned component by manually extracting the relevant model part. When a guarantee needed to be used as assumption for another component, we represented it as a normal SysML block exhibiting all behaviours allowed by the guarantee, again by manually constructing the corresponding state machine. All these steps have been formally verified within our toolbox, thus proving the abstraction correct.

VII. RELATED WORK

In the recent years, several tools have been developed to support model-checking for high-level languages such as UML, SysML or AADL [33]. Based on well known model checkers, they are either directly integrated into tool modellers or they provide transformation techniques from the input model (usually expressed in an XMI file) to the target language of the model checker used. For example, the toolkit SysML Companion [31] provides as extensions model-checking in UPPAAL [39] and Spin [36].

IBM Rhapsody includes validation by simulation in its toolbox. [10] proposes to use UPPAAL in order to verify Rhapsody state machines, the transformation being based on state name equivalence with intermediary states generated for composed actions. System's properties, even though initially described by requirements diagram, have to be given as Computational Tree Logic (CTL) formulas for the UPPAAL model to be checked. However, the lack of an automatized toolset for these transformations makes the approach error prone - the model described in the two frameworks might not be equivalent, which makes harder to identify if the errors found by model-checking are spurious, introduced at the translation, or true system errors.

ARTiSAN Real-time Studio [2] is connected to NuSMV [15] model-checker in [1] which translates a large subset of UML/SysML diagrams (e.g., sequence, activity and timing diagrams, state machine diagrams, block definition and internal block diagrams) to the intermediary Configuration Transition Systems (CTS) formal model. The framework proposes to verify system designs both by model-checking and by metrics and program analysis. Structure diagrams are usually subject to metrics analysis, while behavioural diagrams are translated into the intermediate formal model and model-checked with respect to automatically generated properties and manually specified ones as CTL formulas. For example, hierarchical state machines are transformed into CTS where each state represents either its correspondent one in case of top states or a set of states in case of a

composite state in the original diagram. For each state of the CTS, reachability and deadlock formulas are automatically generated as CTL formulas and verified on the model. However, the temporal extensions for state machines are not presented and neither their impact on the operational semantics. Moreover, state machines are model-checked separately. Our case study is based on the composition of several individual behaviours, composition that is not taken into consideration within this approach which makes it less suitable to be explored. Furthermore, expressing properties as CTL formulas might be a difficult step since system and software engineers are not experts in formal methods and temporal logic.

The Topcased Environment [38] provides connections with UPPAAL and CADP toolkit [11] for SysML models. The AGATE project [32] uses SysML Activity Diagrams, a different formalism than ours, extended with MARTE Profile [22] stereotypes to model the timed behaviour of a system model and translates them into UPPAAL timed automata in order to analyse the execution of the system on a mono platform processor. A second approach within TOPCASED is the usage the pivot language FIACRE [4] to connect AADL and SysML models to CADP. In this case, the latter transformation is not implemented to our knowledge and could not be evaluated.

The academic TTool [37] transforms SysML models that comply with the AVATAR Profile [26] into UPPAAL and CADP for model checking, minimization and bisimulation verification. It takes into account SysML Block Diagrams and state machines extended with temporal operators to express timing constraints (e.g., the *delay* for which the activity of a block is suspended and *computation time* for instructions). The communication between blocks is realized via synchronous and asynchronous channels modelled as connectors (between ports). Safety requirement are expressed using TEPE property language [14], while security properties are expressed and checked within ProVerif [27]. Because TTool is compatible only with the profiles it supports and our project can't be imported, experiments were carried out on toy models. The main difference between the two approaches relates to the sound typing system that OMEGA-IFx provides with respect to ports and connectors. In AVATAR, ports are typeless (contracts cannot be specified for ports), while connectors statically specify which signals they forward and in which direction. This leads to models of bidirectional ports which are a source of typing inconsistencies when translating a SysML model into an implementation language, as explained in [17]. Moreover, a signal can be sent through only one channel and the same channel can transfer requests in both way. This greatly increases the complexity of the verification model, since in our case study the same request is transferred through several connectors; for example, when the four wings communicate their deployment status to the control and monitoring unit, they use the same output port of their owner connected in cascade with the target. Also, the broadcast communication type is not considered in this framework, while OMEGA-IFx implements this option used in industrial system models often implemented in synchronous languages [3]. With respect to the properties model, TEPE uses the parametric diagrams of SysML to express logical and temporal relations between block attributes and signals. The system described by blocks and state machine diagrams is translated into timed automata; parametric diagrams are transformed into observer automata for the UPPAAL model and checked for reachability and liveness. It uses the same principle as our toolbox, with the difference that we express properties as state machines which can be more handy for system engineers than learning a new specification language. One can see than even if AVATAR-TEPE-TTool is an interesting approach, our framework is more industrially-oriented and fits the process of developing safe code for embedded critical systems, also due to the Full MDE Process.

VIII. Conclusions

The difficulties often met during the development of complex critical systems are well-known: late discovery of errors in the system architecture, incompatible interfaces of interacting components, ambiguities in the descriptions leading to misinterpretations by the development teams. The Model Driven Engineering process proposed by the Full MDE project aims at decreasing the number of misunderstandings during the development process, reducing the cost of verification and validation and improving the quality of the final system.

In this paper we have presented the validation of a reverse engineering model of the Solar Generation System of the ATV by using the IFx-OMEGA approach. We have worked on a subsystem developed with OMEGA SysML Profile for which we expressed requirements by observers that were then verified by simulation and model-checking using the IFx toolset. Due to the complexity of the system model the verifications were performed on an abstraction of the model. This framework has allowed discovering residual errors in the system model at early stages of the development process.

The case study has also allowed to point out the weak points of current tools and the path to future improvements. In particular, all proofs were realized on abstraction that have been manually designed. For industrial models, this might be a difficult and time-consuming task. In order to cope with the complexity of nowadays systems, we propose to develop within our framework automated abstractions generation and proofs by integrating a contract-based approach. [30] has proposed a framework for compositional verification of systems with assume/guarantee contracts for different input formalism, like labelled transition systems, modal transition systems [29] or input/output automata [28]. We are currently working on adopting this methodology and extending it for OMEGA SysML components, which are a version of timed input/output automata.

REFERENCES

[1] L. Alawneh, M. Debbabi, Y. Jarraya, A. Soeanu, and F. Hassayne, "A unified approach for verification and validation of systems and software engineering models," in *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, ser. ECBS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 409–418.

[2] Atego ARTiSAN Studio. Available: http://www.atego.com/products/artisan-studio/

[3] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64 – 83, jan 2003.

[4] B. Berthomieu, J.-P. Bodeveix, P. Farail, M. Filali, H. Garavel, P. Gaufillet, F. Lang, and F. Vernadat, "FIACRE: an Intermediate Language for Model verification in the Topcased Environment," in *Proceedings of Embedded Real Time Software and Systems (ERTS2), Toulouse*, 2008.

[5] S. Bornot and J. Sifakis, "An algebraic framework for urgency," *Information and Computation*, vol. 163, 2000.

[6] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," in *Formal Methods for the Design of Real-Time Systems*, ser. Lecture Notes in Computer Science, M. Bernardo and F. Corradini, Eds. Springer Berlin / Heidelberg, 2004, vol. 3185, pp. 131–132.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, january 2000.

[8] E. Conquet, F.-X. Dormoy, I. Dragomir, S. Graf, D. Lesens, P. Nienaltowski, and I. Ober, "Formal Model Driven Engineering for Space Onboard Software," in *Proceedings of Embedded Real Time Software and Systems (ERTS2), Toulouse*, 2012.

[9] E. Conquet, M. Perrotin, P. Dissaux, T. Tsiodras, and J. Hugues, "The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software," in *Proceedings of Embedded Real Time Software and Systems (ERTS2), Toulouse*. SAE, 2010.

[10] E. C. da Silva and E. Villani, "Integrando sysml e model checking para v&v de software crítico espacial," in *Brasilian Symposium on Aerospace Engineering and Applications, Sao José dos Campos, SP, Brasil*, 2009.

[11] J.-C. F. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "CADP - A Protocol Validation and Verification Toolbox," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, R. Alur and T. A. Henzinger, Eds., vol. 1102. Springer, 1996, pp. 437–440.

[12] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, J. v. Leeuwen, J. Hartmanis, and G. Goos, Eds. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996.

[13] J. Hooman, H. Kugler, I. Ober, A. Votintseva, and Y. Yushtein, "Supporting UML-based development of embedded systems by formal techniques," *Software and Systems Modeling*, vol. 7, pp. 131–155, 2008.

[14] D. Knorreck, L. Apvrille, and P. d. Saqui-Sannes, "TEPE: A SysML Language for timed-constrained property modeling and formal verification," in *Proceedings of the UML&Formal Methods Workshop (UML&FM), Shanghai, China*, November 2010.

[15] nuSMV. Available: http://nusmv.fbk.eu/

[16] I. Ober and I. Dragomir, "OMEGA2: A New Version of the Profile and the Tools," in *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*, march 2010, pp. 373 –378.

[17] ——, "Unambiguous UML Composite Structures: The OMEGA2 Experience," in *SOFSEM 2011: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science, I. Cerna, T. Gyimothy, J. Hromkovic, K. Jefferey, R. Kralovic, M. Vukolic, and S. Wolf, Eds. Springer Berlin / Heidelberg, 2011, vol. 6543, pp. 418–430.

[18] I. Ober, S. Graf, and D. Lesens, "Modeling and validation of a software architecture for the Ariane-5 launcher," in *Formal Methods for Open Object-Based Distributed Systems, 2006, Bologna, Italy*, ser. LNCS, R. Gorrieri and H. Warheim, Eds., vol. 4037. Springer-Verlag, 2006, pp. 48–62.

[19] I. Ober, S. Graf, and I. Ober, "Validating timed UML models by simulation and verification," *International Journal on Software Tools for Technology Transfer*, vol. 8, pp. 128–145, April 2006.

[20] I. Ober, S. Graf, Y. Yushtein, and I. Ober, "Timing analysis and validation with UML: the case of the embedded MARS bus manager," *Innovations in Systems and Software Engineering*, vol. 4, pp. 301–308, 2008.

[21] Object Management Group, "Systems Modelling Language (SysML) v1.1," 2008. Available: http://www.omg.org/spec/SysML/1.1/

[22] ——, "Modeling and Analysis of Real-Time and Embedded Systems (MARTE) v1.0," 2009. Available: http://www.omg.org/spec/MARTE/1.0/

[23] ——, "Unified Modelling Language (UML) v2.2," 2009. Available: http://www.omg.org/UML/2.2/

[24] ——, "Object Constraint Language (OCL) v2.2," 2010. Available: http://www.omg.org/spec/OCL/2.2/

[25] ——, "Action Language for Foundational UML (ALF)," 2011. Available: http://www.omg.org/spec/ALF/1.0/Beta2/

[26] G. Pedroza, L. Apvrille, and D. Knorreck, "AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties," in *New Technologies of Distributed Systems (NOTERE), 2011 11th Annual International Conference on*, may 2011, pp. 1 –10.

[27] ProVerif. Available: http://www.proverif.ens.fr/

[28] S. Quinton and S. Graf, "A Framework for Contract-Based Reasoning: Motivation and Application," in *Proceedings of FLACOS'08*, 2008, pp. 77–84.

[29] ——, "Contract-Based Verification of Hierarchical Systems of Components," *Software Engineering and Formal Methods, IEEE International Conference on*, pp. 377–381, 2008.

[30] S. Quinton, S. Graf, and R. Passerone, "Contract-Based Reasoning for Component Systems with Complex Interactions," VERIMAG, Tech. Rep. TR-2010-12, 2010, http://www-verimag.imag.fr/TR/TR-2010-12.pdf.

[31] RealTime-at-Work, "SysML Companion." Available: http://www.realtimeatwork.com/software/sysml-companion/

[32] S. Rochet and Y. Bernand, "AGATE, a transfomation tool for simulation and formal checking for UML/SysML activity (Abstract)," in *TOPCASED Days, Toulouse*, february 2011.

[33] SAE, "Architecture Analysis and Design Language (AADL)," 2004, document No. AS5506/1. Available: http://www.sae.org/technical/standards/AS5506/1

[34] SCADE Suite website, http://www.esterel-technologies.com/products/scade-suite/.

[35] SPARK website, http://www.altran-praxis.com/spark.aspx.

[36] Spin Model Checker. Available: http://spinroot.com/spin/whatispin.html

[37] The TURTLE Toolkit. Available: http://ttool.telecom-paristech.fr/index.html

[38] TOPCASED - The Open-Source Toolkit for Critical Systems. Available: http://www.topcased.org/

[39] UPPAAL v4. Available: http://www.uppaal.com/