Innovations Syst Softw Eng (2011) 7:257–264
DOI 10.1007/s11334-011-0163-2

SI : FM & UML

UML/SysML semantic tunings

Ileana Ober · Iulian Ober · Iulia Dragomir · El Arbi Aboussoror

Received: 30 April 2011 / Accepted: 1 September 2011 / Published online: 2 October 2011

Abstract Recent years have seen a manifest increase in the use of modelling by the embedded systems industry. UML and SysML are two examples of languages used in this context. One of the reasons why the use of models is interesting is the possibility to perform early verification, validation and testing. A lot of work was devoted to developing theoretical results in verification and validation, and interesting results are available. Integrating these results in frameworks that take high-level models as an entry remains a challenging task, for several reasons that include the difficult scalability of the theoretical results. In previous work, we presented OMEGA 2, a framework that takes this challenge. Applying our framework on large industrial models revealed the fact that some features of the UML/SysML semantics which lead to bottlenecks in verification are not actually necessary in the models that we considered, thus leaving place for optimisations. This paper discusses the gap existing between the choices made in the general UML/SysML semantic framework and the actual needs of the users. We illustrate it based on the semantics of ports, for which we give a simplified version of the semantics. This semantics was implemented in our tools and we quantify the optimisation obtained when applying it to a set of case studies.

Ileana Ober (✉) · Iulian Ober · I. Dragomir · E. A. Aboussoror
IRIT, Université de Toulouse, 118, route de Narbonne,
31062 Toulouse, France
e-mail: Ileana.Ober@irit.fr

Iulian Ober
e-mail: Iulian.Ober@irit.fr

I. Dragomir
e-mail: Iulia.Dragomir@irit.fr

E. A. Aboussoror
e-mail: El-Arbi.Aboussoror@irit.fr

Keywords UML · SysML · Composite structure · Operational semantics · Case studies

1 Introduction

Model Driven Engineering approaches are being adopted in a large number of domains. UML [15] and SysML [14] are two examples of languages used in this context. In the area of embedded software, but not only, one of the reasons the use of models is interesting [1] is that they allow for early verification at model level. The early verification is done using various automation techniques, amongst the most used being model checking techniques [5]. One of the major difficulties that model checking has to face is that of its scalability on large models, as they result from current industrial practice. The 2007 Turing Lecture [4] mentioned the need for *scalable* verification methods to cope with "inherent complexity limitations when applied to large systems" and suggested it could be achieved by adapting to "specific semantic domains depending on the data handled". In this paper, we go in this direction and present a semantics optimisation approach. We focus on the semantics of composite structures, as defined in UML 2 and SysML.

In previous work [12], we have presented a UML-based environment for model-checking operational, design-level UML models based on a mapping to a model of communicating extended timed automata. The target language of the mapping is the IF format, for which existing model-checking and simulation tools can be used [3]. This framework was successfully applied on industrial case studies, such as a component of the airborne Medium-Altitude Reconnaissance System [13] and a manually re-engineered model of the Ariane-5 flight software [11]. Recently, we extended the framework to UML 2.0 and SysML, which implies covering

hierarchical composite structure diagrams (or internal block diagrams) [9].

The preliminary applications of this new framework on medium-size case studies were encouraging and were followed by an application on a full fledged industrial case study. Although we were able to pass the first compilation stages, it became obvious that, to be able to perform effective validation, we need to work on optimising the existing semantics. Feed-backs obtained from the case studies suggested some semantics optimisations. We start from the observation that in most of the cases, the notions of port, interface, and connector, present in the case-studies, are much simpler that their complex counterparts defined by the UML 2.0 standard.

The remainder of the paper is organized as follows. We start by a brief overview of the need for hierarchical structures, in Sect. 2, and we present the major characteristics of the hierarchical composite structure diagram semantics, such as defined by UML [15] and covered by our semantics [10]. In Sect. 3, we present some feed-backs obtained when applying "standard" semantics to realistic case studies. These lead us to some semantics optimisations, that we will overview in Sect. 4. In Sect. 5, we evaluate the two semantics, both from a qualitative and a quantitative perspective.

2 Hierarchical composite structure diagrams

In this section, we overview the main concepts present in UML hierarchical composite diagrams (Sect. 2.1) and we make some considerations on their semantics (Sect. 2.2).

2.1 Key concepts

Composite structures are introduced in UML 2.x [2,15] for the specification of "structures of interconnected elements that are created within an instance of a containing classifier". Composite structures are a powerful mechanism to increase the expressiveness and the readability of UML class models. They are used for specifying the initialisation of complex (hierarchical) object structures. This is particularly useful for real-time embedded systems, which are often structured as hierarchical block models, with classes having a fixed number of instances with predefined roles.

An example of a composite structure identifying the model elements involved is given in Fig. 1. A composite structure defines the structure of (instances of) a class in terms of inner components, also called *parts* (`b` in Fig. 1), and of communication *connectors*, also called *links* (`c,d,e,f`). *Connectors* exist either between inner components (`d,e`) or between inner components and the outside environment of the composite structure (`c,f`). A connector has two end points; an end point can be either an inner component or a *port*. A connector can be the realization of an *association*, although this is not mandatory. UML introduces the following terminology for connectors: *delegation* connectors are connectors between a (port of a) part and a port of its containing composite structure (e.g., `c,f`) and *assembly* connectors are links between (ports of) two parts of the same composite structure (e.g., `d,e`).

A *port* represents an interaction point between an object and its environment. The allowed inbound requests that can travel through the port are specified by the *provided interface*(*s*) (e.g., `g`), while the outbound requests are specified by the *required interface*(*s*) (e.g., `h`).

For further technical details, the reader is referred to the UML standard [15].

2.2 Semantic considerations

Composite structures are a big evolution of the object-oriented paradigm, which is the basis of UML. Their implications on the semantic level are huge and not completely defined by the UML standard (see for example the problems outlined in [6,10]).

The principle of the OMEGA profile is to have a clear and coherent executable semantics for the chosen UML subset, i.e. conform to the UML standard. In previous work, we presented the semantics of the composite structure diagrams [10]. We focus here only on the elements used in the context of this paper: the semantics of ports.

For the translation of composite structures, the main challenge was to fit the relatively complex, hierarchical UML modelling constructs into the simple and flat object space of IF automata, while remaining as close as possible to the UML standard semantics. While remaining rather abstract, the UML standard semantics gives some information on the expected semantics. For instance, with respect to a port it states that: "when an instance of a classifier is created, *instances corresponding to each of its ports are created* and held in the slots specified by the ports, in accordance with its multiplicity".

Based on this, we built a semantics using the principle that the ports should be handled—at the semantics level—as first class language citizens. Concretely, each port instance is implemented as an IF process instance and each connector is represented by attributes in the end points (in ports or in components). A UML composite structure diagram is thus used as an initialisation scheme for instantiating components and ports and for creating links. The composite structure is, therefore, translated to a constructor (see [12] for a description of constructors in OMEGA).

The tests we performed on this semantics proved that the semantics choices we made are sound and we were able to catch errors in the composite structure of some models. Preliminary medium size tests performed on this semantics, in terms of model-checking the IF translation of realistic

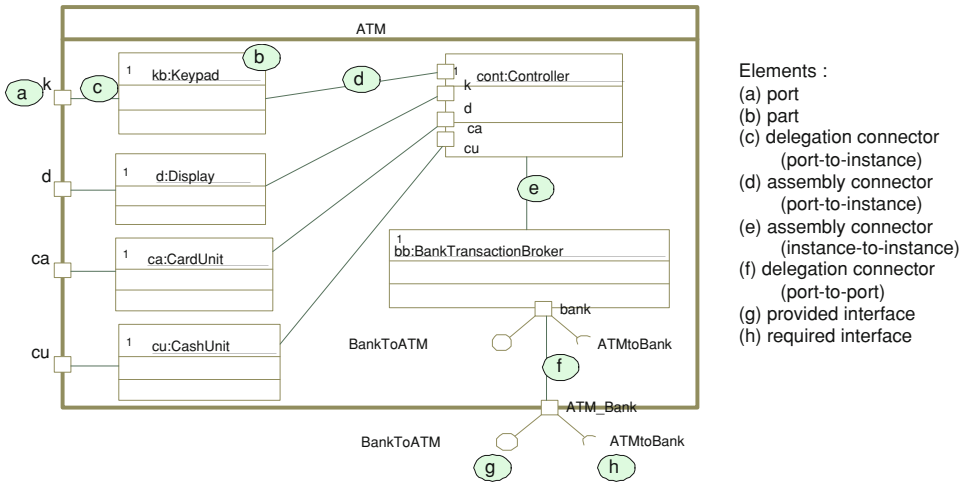

Fig. 1 Example of a composite structure diagrams with its main concepts

models using complex composite structure diagrams, provided encouraging results and we went further and applied our semantics on full-fledged industrial case studies.

3 Feed-back from industrial case studies

In the context of the FullMDE project, we applied the OMEGA 2 toolset on a set of small toy examples as well as one large industrial case study, the solar generation system (SGS) of the automated transfer vehicle (ATV)[1] designed by Astrium Space Transportation.[2] The concrete specification of this system as well as the verifications performed on these models are out of the scope of this paper. However, we will stop on some lessons that emerged from these case studies.

The main objectives of this experiment were the following:

1. to assess whether the OMEGA profile is expressive enough to cover all the situations that can be encountered in real case studies;
2. to make sure the semantic choices are coherent with the user expectations;
3. to see how far we can go in applying verification techniques on these models.

An important feature of this experiment is that the models were conceived independently by the field engineers after a minimal training on using our tool suite. This is an important feature because the end users are the ones that can best assess the expressiveness of a language, with respect to their needs. It implies that the modelling framework should be usable, without deep knowledge of the technicalities behind each concept.

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

[2] http://www.astrium.eads.net.

The hierarchical structure diagrams proved to be a central artefact for the system models. In our case studies, the users intensively used structure diagrams to cope with complexity. For example, the SGS case study contains 25 internal block diagrams (IBD) at various nesting levels in the model (there are 4 levels of block containment), each diagram containing up to 13 parts and 68 connectors. In some cases, several IBDs are used to describe the same block from different view points, resulting in blocks that have up to 14 parts and around 130 contained connectors.

The OMEGA2 tools introduce a set of rules for constructing sound and well-typed composite structures, which are detailed and rationalized in [10]. The overall conclusion from the case studies is that the rules are not over-restrictive and went mostly unperceived, as they parallel best practices rules adopted by the engineering team. Concerning the operational semantics of structured classifiers, however, the choices that were made initially were sometimes challenged by observations made during the case studies, as we show in the sequel.

One of the first observations that we were able to make was that, often, UML/SysML offers several alternative ways for encoding the same information. While this is a frequent source of problems for tool builders and semantics definitions, we should turn it into an advantage, by inciting the users to make the choices that are closer to their particular application domain and to make sure the tools support this.

For instance, in SysML there are several ways to specify connectors between blocks. A connector can be linked to a block instance either directly or via a port. For a directly linked connector to be meaningful, it is in general necessary to type it with an association (the exact well-formedness rules are defined in [10]). However, during the experimentation phase, we found out that system engineers neither use directly linked connectors, nor use associations, which are viewed as a software-level concept with weak semantics and

not suitable for system-level modelling. Instead, ports with clearly identified provided/required interfaces are systematically used. This type of observation can help prioritise the features offered by a tool and provide optimisations for the most relevant concepts from a user perspective.

Another observations are that, sometimes, *the user's demands for features go beyond their actual needs and are irrespective of the induced price*. As a concrete example, during the preliminary discussions, the users expressed their need to have a *broadcast* semantics for ports that have multiple connectors transporting the same interface departing from them. (Alternatives, also used in other tools such as IBM Rhapsody[3] or Tau,[4] would be to either choose non-deterministically the connector through which requests are delivered, to signal this as a modelling error, or to demand that the port behaviour be explicitly specified in that case by the designer.) During the case studies, however, it turned out that this functionality was never used, since it diverges from the functioning of the deployed systems. Still, offering the possibility to do broadcasts induces overhead in the underlying semantics that affects even the case studies not using this feature. For example, in the case of a broadcast, the message received by the port of a composite class needs to be dispatched to all the parts that are able to treat it and need to be computed.

Finally, the most important observation concerns the behaviour of ports. UML/SysML offers a very powerful concept of port, which can have an associated behaviour or protocol state machine. Nevertheless, none of the 388 ports in the SGS model actually used this feature. Across the case studies, the ports were always used as simple interaction points, having the default behaviour of transferring requests from a source to a destination connector according to their type. Despite that, all these ports lead to new active entities at the level of the semantics, which is a significant unnecessary overhead, as we will show in the next sections.

This leads us to the conclusion that a quantified evaluation of the costs induced by choices between alternative semantics would allow more substantiated decisions, closer to the actual needs of the users. The next two sections concentrate on one such example of semantic tuning: we present a lighter alternative semantics for ports and connectors and we make a qualitative and quantitative evaluation with respect to the standard semantics. The outcome of these evaluations is that, under certain conditions and in well-identified contexts, a lighter semantics could be used. The standard semantics remains obviously the one capable of dealing the variety of situations the can be encountered in practice.

[3] http://www.ibm.com/software/awdtools/rhapsody.

[4] http://www.ibm.com/software/awdtools/tau/.

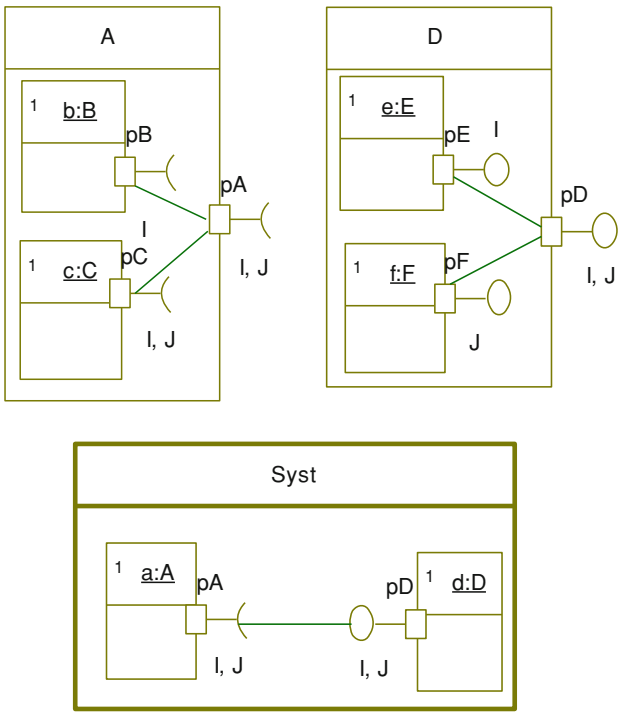

Fig. 2 Hierarchical composite structure example

4 An example of semantic tuning

In this section, we present the technical details of a semantics simplification example, which addresses the issue of port behaviour. As we have seen above, although it does not give a complete semantics for ports, the UML standard implies that ports are active entities and this is also the way they were implemented in OMEGA.

To illustrate this, we use the hierarchical composite structure example in Fig. 2. B, C, E and F are active classes (owning behaviour described by state machines), but with no hierarchical structure of their own. We do not give here their complete definition, as it can be either inferred from what is already present in the upper part of Fig. 2 (in terms of ports and implemented/required interfaces), or it is not important for the purpose of this example.

A is a class that contains two parts—an instance of the active class B, respectively an instance of class C—and a port pA. In the context of A, the "wiring" of ports and connectors occurs as illustrated in Fig. 2. Similarly, D is a class containing parts, ports and connectors as illustrated in the figure.

The upper part of Fig. 2 gives the type definitions of A and D. The lower part of the figure gives an example of their possible combined usage in the context of another composite class (here Syst).

Note that in other parts of the model, any of these classes could be used in different ways, provided their port constraints

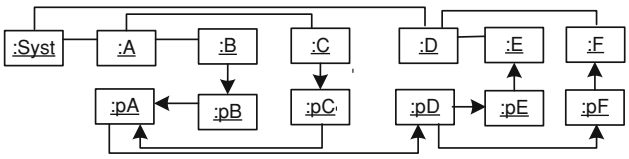

Fig. 3 Object diagram according to the original semantics

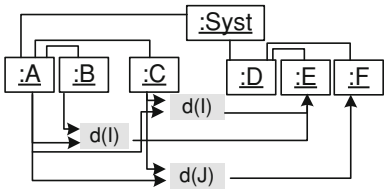

Fig. 4 Elements in the simplified semantics

are satisfied. The structures shown in Fig. 2 are used at run-time to create instances of composite classes (A, D, Syst) by creating and wiring the respective sub-components.

The run-time semantic structure corresponding to the example, as implied by the UML semantics, is presented in Fig. 2. Each time the Syst class is instantiated, it results in the creation of the elements described in the object diagram presented in Fig. 3. In this object diagram, we have instances corresponding to each of the parts present in Syst, i.e. one instance of the class A and D respectively, as well as instances for each of their ports (here pA and pD). The creation of the instances of A and D leads to the recursive initialisation of their respective structure (B,C, pB, pC, …). Figure 3 describes the connectors existing between these instances as well as their directionality. Their creation is also part of the semantics, and enables the communication as specified in the definition of class A.

The entities generated corresponding to the initialisation of A as well as those generated corresponding to the initialisation of D are respectively connected, as specified in the definition of Syst, and result in the connector that binds pA and pD.

One can see that the semantics of Syst is given in terms of 12 active entities, out of which six correspond to ports. Our semantics being expressed in terms of a process-based formalism (IF, see [3]), each of these entities results in the creation of a new process. The behaviour of the processes corresponding to the ports (pA, pB, …, pF) is fairly simple and it consists of systematically forwarding the received messages, unless advanced port policies are expressed at the level of the UML model in terms of state machines attached to ports.

The complete absence, across our case studies, of state machines attached to ports implies a possible optimisation: replacing the objects corresponding to ports by passive (data only) entities. There is, however, a difficulty in eliminating the objects that correspond to ports, which comes from the completely dynamic nature of the model: in our example, the instantiation of C results in a part of A that forwards the messages to the context embedding A. However, C may also be instantiated in other contexts, and one cannot statically determine the destination of C's messages, as it is depending on the hierarchical structure in which the instance of B is embedded. It is obvious that the actual destination of the messages sent out by C can only be calculated at run-time and its type may vary depending on the structure that encloses C.

Note that we aim for a *modular* compilation method, i.e. a method that allows to compile the block C in a unique way regardless of the contexts in which C is instantiated. This means that the code corresponding to C cannot contain any direct information of the destination of the messages, but rather use an indirection mechanism. The mechanism has to be flexible enough so that the initialisation of the actual connections of a composite (e.g., an instance of A) does not need to involve knowledge about the inner components (b, c).

In the interest of space, we do not give here all the details of this alternative translation, but instead we explain how it works on the UML model sketched in Fig. 2. As in the case of the "classical" semantics, the active objects result in active entities. This leads to active entities corresponding to the instances of A, B, C, D, E and F, as shown in Fig. 4. These are the only active entities in our semantics, since the ports are modelled as passive data.

Let us examine in which order the various elements present in Fig. 4 are created. Based on the composite structure diagrams described in Fig. 2, at the instantiation of *Syst* are created one instance of A and one instance of D, as well as the connector between pA and pD. The creation of A involves to the creation of one instance of B and one instance of C. At the creation of C, due to the presence of *pC*—that requires the interfaces *I* and *J*—two passive objects of a particular type called *routing cell* are created, which will eventually (i.e., once the connections of A will be initialised) store the destinations for requests emanating from pC and belonging to I and respectively J. A *routing cell* is a very simple passive object which stores the destination information for a particular (port,interface) pair (in IF, the destination is simply represented by a process identifier).

The routing cells are represented in Fig. 4 as grayed rectangles (*d(I)* and *d(J)*). The creation of B also leads to a routing cell for the required interface I (also denoted *d(I)*). Once b and c have been created, the structure diagram of A leads to the creation of pA and of the connectors: pB to pA and pC to pA. The semantics of these two UML connectors is that messages sent out via pB and pC get to the future external connection of pA. As a result, pA will contain a reference to the routing cells *d(J)* of C and to the *d(I)* of B and C, as shown in Fig. 4.

Creating D, leads to the creation of one instance of E and one instance of F. Moreover, based on its composite structure, D's port variable contains references to the instances E and F.[5] Finally, after the instances of *A* and *D* are created, the connector *pA* to *pD* is created. This consists of filling the routing cells pointed by *pA* with the values referred by *pD*. Note that this can be done without knowledge of the connections and subcomponents of A and D.

5 Evaluation

5.1 Qualitative evaluation

Two important questions arise for every semantic tuning:

1. what restrictions to the models does the semantic tuning incur?
2. in what cases the alternative semantics is equivalent with the original semantics and in what cases it is not?

These questions cannot be given a general answer and can only be treated case-by-case. The first question is generally simple and the answer derives from the definition of the semantic tuning. For instance in our example, the restrictions are that models should only use standard ports (no flow ports), not use user-defined port behaviour nor broadcast ports.

The second question is harder and the equivalence argument is difficult to formalize. For our example, it became clear after working with a few specifications that the semantics of a system is preserved if there is no possibility for message overtaking along multi-channel connections in the original model and if the model abstracts away from delays in ports/channels. This is due to the nature of our routing scheme, which propagates the routing information when components/links are created, and then allows a signal to be taken from source to destination in only one step. If for example b and c emit an I signal each (in this order), the signals will arrive in the same order at the destination (e). In the original UML/SysML semantics, it is possible that the channel pB-pA is slower than pC-pA, and the signal from c overtakes the one from b. We found, however, that the kind of high-level system models that are usually handled with OMEGA is rarely concerned with message overtaking and channel/port congestion (as specific performance analysis methods are usually used for this).

[5] In this semantics, we do not consider broadcast ports, which implies that at the level of pD there is only one connection conveying signals belonging to I and only one for signals belonging to J. Note that it is possible to extend the mechanism to support broadcast ports. As this incurs an overhead, it should be treated as a semantic tuning and should be balanced according to the need for such broadcast ports in actual models.

When both the standard and the alternative semantics can be used to generate a manageable behaviour model of a system, in form of a labelled transition system (LTS), one can prove the equivalence of the two semantics (modulo hiding of port actions) using LTS equivalence. As an example, we have tried this on a medium-size model of an automated teller machine (ATM—see next section) and have been able to prove that the two semantic models are strongly bisimilar, using the CADP toolbox [7]. Since the point of semantic tuning is to simplify the semantics so that larger models can become tractable, it is in general not possible to do this comparison on all the models for which the simplification is necessary. In such cases, it is useful to have a set of test models on which semantic equivalence can be tested with the method mentioned above to gain confidence in the faithfulness of the semantic tuning.

5.2 Quantitative evaluation

In this section, we make a quantitative assessment of the simplification of port semantics presented in Sect. 4, in the interest of showing the potential gains for such semantic tunings. The quantitative evaluation is important for a semantic tuning, as it gives the user the possibility to choose what semantic to use for a particular model, by making trade-offs based on her needs.

We have comparatively applied our two semantics for ports on the two case studies mentioned before: the toy model of an ATM and the real-world model of the SGS. To characterise these case studies and to give an idea on their complexity, we use the following metrics:

- Number of classes/blocks (NBlocks);
- Number of ports (NPorts);
- Number of connectors (NConn);
- Depth of the hierarchical structure (DPT);
- Maximum number of ports per class/block (MaxPorts);
- Average number of ports per class/block (AvgPorts);
- Maximum number of connectors connected to a port (MaxCP);
- Average number of connectors connected to a port (AvgCP).

Table 1 gives a characterisation of the two case studies with respect to the metrics introduced above.

To assess the impact of using the two alternative semantics, we analyse the resulting IF code, as it results when applying the original semantics and the tuned semantics.

In the case of the ATM case study, we analyse the following measures:

- *number of processes (NoP)* after the translation of the specification in IF. An important number of IF processes corresponds to an important number of active entities running in parallel, leading to extra complexity.

Table 1 Metrics characterisation of the two case studies

	ATM	SGS
NBlocks	8	24
NPorts	19	388
NConn	15	397
DPT	3	4
MaxPorts	6	106
AvgPorts	4	16
MaxCP	3	8
AvgCP	2	2

Table 2 Comparative analysis in the case of the ATM

	NoP	NoS	NoS_{POR}
Standard semantics	55	11998	5900
Alternative semantics	33	5552	2883
Reduction (%)	60	46	49

Table 3 Comparative analysis in the case of the SGS

	NoP	$NoSteps$
Standard semantics	984	5461
Alternative semantics	422	2350
Reduction (%)	43	43

- *number of states (NoS)* gives the size of the state space.
- *number of states after partial order reduction* (NoS_{POR}) gives the number of states that remain explored when applying the partial order reduction algorithm of the IF model-checker.

Table 2 gives the values for these measures. According to these results, the change in semantics leads to reducing the number of active entities (IF processes) to approximately 60%. This active entities reduction results in an important reduction of the number of states (NoS) to 46%. The reduction is approximately the same when applying partial order reduction, implying that the two reductions (semantic tuning and POR) are additive.

The metrics for the SGS case study are different, since SGS describes a much more complex behaviour and the complete state space cannot be generated without further semantic tuning. However, as its behaviour is acyclic, one can look at the impact of the semantic tuning on the size of a nominal scenario (i.e., from start-up to the full deployment of the solar wings). The *number of steps* ($NoSteps$) measures the length of the nominal scenario. Table 3 gives a quantitative comparative analysis of NoP (defined as before) and $NoSteps$, in the case of the two alternative semantics.

These measurements show that one can quantify the cost of a particular semantic choice. This has the advantage that it allows the users to make informed choices with respect to the use of certain constructs or semantic "profile". On the other side, if this method is taken to the extreme, it may have the drawback that users may have the tendency to use too primitive concepts, leading to models that are harder to test, maintain and extend.

6 Conclusion

The interest of applying formal analysis methods as early as possible in the development life-cycle has been often established [8]. However, applying these techniques on complex models is challenging. A lot of work has been dedicated to dealing with complexity and combinatorial explosion, e.g., in the model-checking community (see [4] for a survey). Using an over-complex semantics may impact other analysis methods too, including simulation and testing. In this paper, we argue that it may be useful to work on adapting the semantics based on the characteristics and needs of the models that are considered. To support this point, we consider one particular semantic optimisation in the context of UML/SysML composite structures, which was implemented in the OMEGA2 framework, and for which the gains are quantified on the basis of two case studies.

Beyond the semantic optimisation example, this paper argues on the need to analyse the correlation between the semantics of the model and the actual needs of its application domain. General languages, such as UML, have the huge advantage of being largely known and applicable. However, they often bring an overhead that proves to be critical for model-based verification and validation activities. For the communities using these languages, we think that it may be interesting to set up a repository of semantic variants, which makes explicit the details about the induced restrictions, the sufficient conditions for equivalence with the standard semantics and the potential gains in terms of various metrics, along the lines of what we did for the example considered in this paper. Note that in our examples we focused only on the potential benefit for simulation and model-checking. A similar reasoning may be applied with respect to other parameters such as testability, maintainability or compatibility with the characteristics of the application domain.

The existence of such a repository would make possible to create customised semantics, based on the application (domain) needs. This would also open the way for better knowledge of the advantages and drawbacks induced by the choice of particular concepts or semantic variations.

References

1. Balasubramanian K, Krishna AS, Turkay E, Balasubramanian J, Parsons J, Gokhale AS, Schmidt DC (2006) Applying model-driven development to distributed real-time and embedded avionics systems. Int J Embed Syst 2:142–155

2. Bock C (2004) UML 2 composition model. J Object Technol 3(10):47–74
3. Bozga M, Graf S, Ober I, Ober I, Sifakis J (2004) The IF toolset. In: Bernardo M, Corradini F (eds) Formal methods for the design of real-time systems. In: International school on formal methods for the design of computer, communication and software systems, SFM-RT 2004, Bertinoro, Italy, September 13–18, 2004, Lecture notes in computer science, vol 3185. Springer, Berlin, pp 237–267
4. Clarke EM, Emerson EA, Sifakis J (2009) Model checking: algorithmic verification and debugging. turing lecture from the winners of the 2007 ACM A.M. Turing Award. Commun ACM 52(11):74–84
5. Clarke EM, Grumberg O, Peled DA (1999) Model checking. MIT Press, Cambridge
6. Cuccuru A, Gérard S, Radermacher A (2008) Meaningful composite structures. In: Czarnecki K, Ober I, Bruel J-M, Uhl A, Völter M (eds) MoDELS. Lecture notes in computer science, vol 5301. Springer, Berlin, pp 828–842
7. Fernandez J, Garavel H, Kerbrat A, Mounier L, Mateescu R, Sighireanu M (1996) CADP—a protocol validation and verification toolbox. In: Alur R, Henzinger TA (eds) CAV. Lecture notes in computer science, vol 1102. Springer, Berlin, pp 437–440
8. Hinchey M, Jackson M, Cousot P, Cook B, Bowen JP, Margaria T (2008) Software engineering and formal methods. Commun ACM 51(9):54–59
9. Ober I., Dragomir I (2010) OMEGA2: a new version of the profile and the tools. In: Calinescu R, Paige RF, Kwiatkowska MZ (eds) ICECCS. IEEE Computer Society, Washington, pp 373–378
10. Ober I., Dragomir I (2011) Unambiguous UML Composite Structures: The OMEGA2 experience. In: Cerná I, Gyimóthy T, Hromkovic J, Jeffery KG, Královic R, Vukolic M, Wolf S (eds) SOFSEM. Lecture notes in computer science, vol 6543. Springer, Berlin, pp 418–430
11. Ober I, Graf S, Lesens D (2006) Modeling and validation of a software architecture for the Ariane-5 Launcher. In: Gorrieri R, Wehrheim H (eds) Formal methods for open object-based distributed systems. In: Proceedings of the 8th IFIP WG 6.1 international conference, FMOODS 2006, Bologna, Italy, June 14–16, 2006, Lecture notes in computer science, vol 4037. Springer, Berlin, pp 48–62
12. Ober I, Graf S, Ober I (2006) Validating timed UML models by simulation and verification. STTT 8(2):128–145
13. Ober I, Graf S, Yushtein Y, Ober I (2008) Timing analysis and validation with UML: the case of the embedded MARS bus manager. ISSE 4(3):301–308
14. OMG (2008) Object Management Group—systems modeling language (SysML), v1.1. http://www.omg.org/spec/SysML/1.1/
15. OMG (2009) Object Management Group—unified modeling language (version 2.2). http://www.omg.org/spec/UML/2.2