# Integrating Verifiable Assume/Guarantee Contracts in UML/SysML

Iulia Dragomir, Iulian Ober, and Christian Percebois

Université de Toulouse - IRIT
118 Route de Narbonne, 31062 Toulouse, France
{iulia.dragomir,iulian.ober,christian.percebois}@irit.fr

**Abstract** The compositional approach based on components and driven by requirements is a common method used in the development of critical real-time embedded systems. Since the satisfaction of a requirement is subject to the composition of several components, defining abstract and partial behaviors for components with respect to the point of view of the requirement allows for a manageable design of systems. In this paper we consider such specifications in the form of contracts. A contract for a component is a pair (assumption, guarantee) where the assumption is an abstraction of the component's environment behavior and the guarantee is an abstraction of the component's behavior given that the environment behaves like the assumption. In previous work we have defined a formal contract-based theory for Timed Input/Output Automata with the aim of using it to express the semantics of UML/SysML models. In this paper we propose an extension of the UML/SysML language with a syntax and semantics for contracts and for the relations they must satisfy. Besides the important role that contracts have in design, they can also be used for the verification of requirement satisfaction and for their traceability.

## 1 Introduction

Nowadays critical real-time embedded systems grow larger in size and more complex. Their development is a challenging task and is often error-prone. A way for system designers to tackle this issue is to use a compositional approach driven by requirements. For example, process-oriented standards such as DO-178C [14] highlight the need to model requirements at different levels of abstractions during development and to ensure their traceability at each design iteration step.

However, requirements are often difficult to be mapped to components: several components combine together to satisfy a requirement and a component may be involved in the satisfaction of several requirements. In order to achieve provably correct compositional design, one needs a way to abstractly specify how a particular component $K$ participates in fulfilling a requirement $\varphi$. Such a specification can take the form of a *contract*: a pair (*assumption*, *guarantee*) where the *assumption* is an abstraction of $K$'s environment behavior and the *guarantee* is an abstraction of $K$'s behavior given that the environment behaves according to the assumption. Such a contract can then be used to model the point of view of the component with respect to the requirement $\varphi$. Contracts for reactive and real-time components have received a lot of attention from the research community recently, as discussed in §5.

Besides the important role contracts can play in system design, they can also be used as basic blocks for compositional verification of requirement satisfaction. In [11] we have introduced a contract-based theory for compositional verification of systems of communicating Timed Input/Output Automata (TIOA) with the intention to use it as underlying semantics for contract-based UML/SysML [19, 18] modeling and verification.

This paper complements the theory from [11] by extending UML/SysML with the language elements needed for modeling contracts and their relations. The paper

is structured as follows: in §2 we summarize the contract-based reasoning theory we have defined and we present the OMEGA UML/SysML profile [8] on which we want to apply the formal theory. In §3 we propose a meta-model for the contract theory and a set of constraints and well-formedness rules needed to make the system model verifiable with contracts. Then, an instantiation of the meta-model for the OMEGA UML/SysML profile is discussed. §4 presents the application of our approach to an industry-grade system model, the ATV SGS case study previously described without contracts in [9], before concluding.

## 2 Background

### 2.1 Timed Input/Output Automata

Many mathematical formalisms have been proposed in the literature for modeling communicating timed reactive components. Our work is based on a variant of Timed Input/Output Automata of [13] since it is one of the most general formalisms, thoroughly defined and for which several interesting compositionality results are already available. A TIOA specifies a state space and a set of admitted timed behaviors for a component. The parallel composition of TIOAs (denoted $\|$ in the following) is based on synchronization of corresponding inputs/outputs and the interleaving of other actions. The main differences between our variant and that of [13] are that (1) matched input-outputs resulting from a composition of two automata result in a visible I/O action which is not involved in synchronizations thereafter (in [13] the result is an output, which is not consistent with the semantics of signals in UML or SysML) and (2) continuous variables are restricted to Alur-Dill-style linear clocks [1] in order to allow the use of known symbolic simulation and verification methods (unavailable for the more general model of [13]). For the full definition of the formal model, the reader is referred to [11].

### 2.2 A UML/SysML Profile Based on Timed Input/Output Automata

In previous work [5] we have considered the high-level modeling of embedded real-time systems in UML/SysML with a semantics provided in terms of TIOA. The result of this work is a semantic profile for UML and SysML called OMEGA and a set of tools for simulation and model-checking, the IFx-OMEGA toolset[1].

In OMEGA, the architecture of a system is expressed in the usual way in UML (class diagrams) and in SysML (block definition diagrams and internal block diagrams). Classes/blocks may use most of their features: properties (attributes and parts), signals receptions, interconnection elements (port, connector, interface) and relations (association, composition and generalization). The hierarchical architecture of components (and systems) is specified through composite structures.

The behavior of atomic components is modeled by state machines with standard UML actions on transitions. The operational semantics of each component instance is a timed input/output automaton. The TIOAs corresponding to components are composed in parallel and communicate by asynchronous signal exchanges. This imposes that all communications between objects/block instances are defined as signal outputs and receptions that are transferred via ports and connectors. Ports need to be typed with interfaces that contain the list of signals transferred to or from the component's environment. With respect to the actions described on transitions, the semantics supports a significant part of the UML action meta-model, including signal output, assignment, expression valuation and control flow structuring statements. The translation between OMEGA modeling concepts and the underlying TIOA semantics is automatic in IFx-OMEGA.

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

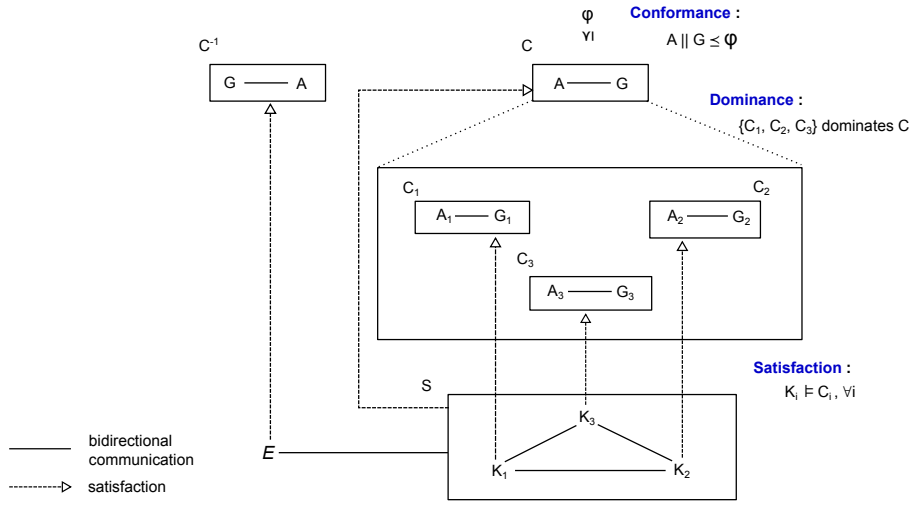


Figure 1: Contract-based reasoning for a subsystem with three components [21].

The temporal elements of TIOA, such as clocks, clock actions and timed guards actions have corresponding language constructs in the OMEGA profile. The elapsing of time is constrained by *transition urgency* stereotypes inspired from timed automata with urgency [4]: time delay is *blocked* if one of the active transitions in the current state is stereotyped «*eager*», it is *upper-bounded* if an active transition is stereotyped «*delayable*» and is unbounded otherwise (i.e., if all active transitions are stereotyped «*lazy*», which is by default).

The profile also proposes mechanisms for formalizing requirements, in particular in the form of real-time safety properties described by *observer* classes (identified by a stereotype «*observer*»). The state machine of these classes uses special primitives for monitoring the system state and events and gives verdicts about the (non-)satisfaction of a property by using labels (e.g., stereotype «*error*») on states.

For a more complete description of the UML/SysML component model used in OMEGA and of the mapping of notions, the reader is referred to [17].

### 2.3 A Theory of Contracts for Timed Input/Output Automata

In [11] we defined a theory for modeling and reasoning with contracts for TIOA. The theory is an extension of a meta-theory defined in [22, 21]. A contract for a TIOA component $K$ is a pair of TIOAs that model the assumption ($A$) and the guarantee ($G$). The satisfaction of the contract is defined formally by a relation based on trace inclusion between $K \parallel A$ and $G \parallel A$ modulo the set of actions that are of interest for the contract. The theory also provides the necessary mechanisms for compositional reasoning with contracts, explained in the following.

Consider that the objective is to prove that a system $S$ composed of several components $K_1, K_2, \cdots, K_n$ satisfies a property $\varphi$ (see Fig. 1) under a certain hypothesis $A$ on the behavior of its environment. The method consists in defining a more abstract specification $G$ of the system such that $A \parallel G$ satisfies $\varphi$ (*conformance* step in Fig. 1). Once this step performed, one will be inclined to verify that the system $S$ satisfies the contract $C = (A, G)$. However, it is often impossible to verify directly (i.e. by checking trace set inclusion) that the composite system $S$ satisfies the contract because of the combinatorial explosion of the state space. To avoid this problem, the method defined in [11, 21] uses a decomposition of the proof in independent steps based on the definition of a set of individual contracts $C_1, C_2 ...$ for

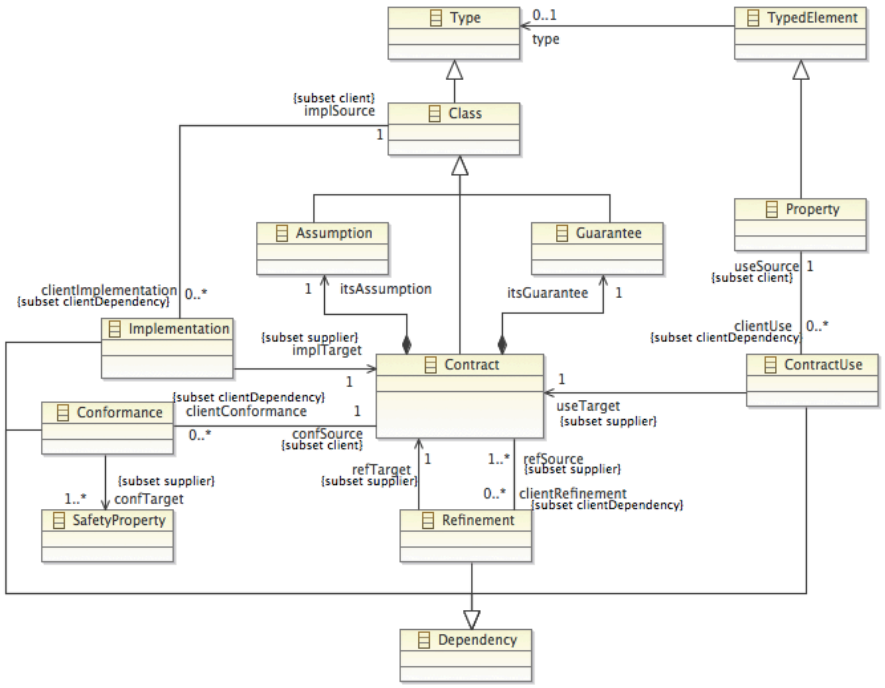


Figure 2: UML meta-model extended with contracts.

the components $K_1, K_2$..., which, when put together, ensure the global contract $C$. We say that $\{C_1, C_2, \cdots\}$ *dominates* $C$ (*dominance* step in Fig. 1). The theory in [11, 21] provides a set of *sufficient conditions* for dominance which can be checked independently with lesser combinatorial complexity. In addition to the conditions for dominance, one also has to check that each component $K_i$ satisfies the contract $C_i$ (*satisfaction* step in Fig. 1).

## 3 Extending UML and SysML for Modeling Contracts

In this section we present the UML and SysML extensions that we propose in order to support modeling and reasoning with contracts. In §3.1 we describe a domain meta-model of the contract-related concepts which extends the UML meta-model. The adaptation for SysML is discussed briefly at the end, as is relatively straightforward. We then discuss in §3.2 the constraints and well-formedness rules imposed on the key notions in order to make models with contracts compliant with the theory from [11] and thus verifiable. Finally, in §3.3 we discuss the mapping of the meta-model concepts as a UML/SysML profile, using the standard extension mechanisms (stereotypes).

### 3.1 A Meta-Model for Contracts in UML

Within the contract theory we have presented there are two categories of concepts: (1) those related to contract modeling represented in the upper part of the meta-model given in Fig. 2 and (2) those related to modeling relations between contracts, used for example in verification, represented in the lower part of Fig. 2.

The requirement $\varphi$ that the component model has to satisfy is represented by the meta-class *SafetyProperty*. This meta-class is left unspecified at this point since

different formalisms could be used to model a requirement, such as temporal logics, automata-based languages, etc. In §3.3 we instantiate the meta-model in the OMEGA profile, which uses an observer for modeling a *SafetyProperty*.

The assumption/guarantee of a contract is modeled by the corresponding meta-class *Assumption*/*Guarantee* type of *Class*. Each of these two elements is modeled by a class that has a behavior expressed by a state machine and communicates (only) through ports. Assumptions/guarantees are thus described in the same language as the system components. In order to have a clear semantics in terms of TIOA, they are subject to few restrictions: they should not be involved in associations, generalizations, realizations (except the interface realizations demanded by ports) and dependencies. An assumption or guarantee may define a composite sub-structure. Such an example is provided in the case study of §4.

A contract is represented by the meta-class *Contract* as a composite structure, containing exactly one *assumption* and one *guarantee* (i.e. any other properties are forbidden) and does not exhibit any behavior. The only relations a contract may be involved in are those that represent the verification relations used in our theory, as described below.

If a contract serves in the conformance step in the methodology depicted in Fig. 1, this is modeled using a *Conformance* relation (a kind of *Dependency*) between the *Contract* and the corresponding *SafetyProperty*. One can use the same contract for several safety properties.

The dominance relation is represented by the meta-class *Refinement* type of *Dependency*. One contract is *refined* by a set of contracts. Note that this is possible since UML defines *Dependency* from *n clients* to *n suppliers*. To ensure that no cycles may be modeled, the following constraint is imposed: the target of a *Refinement* is not a member of the source set.

Finally, the relation between a component and the contract that it must satisfy is represented by two relations: one at the level of the type of the component and one at the level of the instance (the part which participates in a composite structure where the contract is relevant and which is modeled by the meta-class *Property*). On the level of the type, an *Implementation* relation (a kind of *Dependency*) between a class and a contract models the fact that the class satisfies the contract. One class can satisfy several distinct contracts. On the level of instances, a *ContractUse* relation (also a kind of *Dependency*) between a *Property*, which is part of a composite structure, and a *Contract* models the fact that the contract is used for verification within the context of that composite structure. A *Property* may *use* a contract if and only if its class *implements* that contract.

All constraints have been formalized with OCL [20]. Their code can be found in an extended version of this paper [10]. This meta-model can be easily adapted for the SysML language by defining the meta-classes *Contract*, *Assumption* and *Guarantee* as extensions of the stereotype *block* applied on the meta-class *Class* from *UML4SysML* package. Similarly, the meta-class *Dependency* belongs to the same package.

### 3.2 Well-Formedness Rules for Verifiable Contracts

In order to be able to apply the contract-based verification theory from [11] we need to make sure that the hypotheses and constraints imposed by the formal framework are satisfied by the system model. In the following we formalize these constraints at the meta-model level by a set of well-formedness rules.

Within the formal framework, a contract is modeled by a pair $(A, G)$ of TIOA such that the set of inputs/outputs of $G$ is a subset of the set of inputs/outputs of the component implementing the contract and the composition of $A$ and $G$ is a closed system. To ensure this, the set of ports of a *Guarantee* must correspond to a subset of the set of ports of the component for which the guarantee is defined. The correspondence is based on the port name and the port type and direction

must coincide. We consider that when a port is present in the guarantee, all the corresponding signal receptions defined by the port type are handled in the guarantee. Moreover, as the composition between an *Assumption* and a *Guarantee* must be closed, every port of the *Guarantee* must have a corresponding conjugated port on the side of the *Assumption*, with the same type and reversed direction.

The dominance relation is also subject to the refinement of provided/required requests. This rule is also expressed with respect to ports: a port of the guarantee which is the target of the refinement must be matched (by name and type) by a port of one of the refining guarantees and must not be matched by a corresponding conjugated port (i.e. with reversed directionality) of another from the refining guarantees.

The theory from [11] also induces some constraints on the state machines of assumptions and guarantees. In particular, the behavior of an assumption or guarantee should not impose constraints on time progress. This is realized on the UML/SysML level ensuring that all transitions in these state machines are stereotyped «*lazy*» and that there is at most one output action on any transition.

Furthermore, for a model with contracts to be used in compositional verification according to the methodology described in §2, the model must describe a unique and complete proof tree: all implemented contracts are used within a context and for all *SafetyProperty* there is a contract conforming to it. The rules described above have been formalized in OCL and can be found in [10].

### 3.3 Instantiating the Meta-Model in the OMEGA Profile

In order to use contracts in a standard UML or SysML model, one needs to capture the information from the meta-model described in the previous section in the form of standard extensions, namely using stereotypes. Since all the new concepts introduced in the meta-model inherit from an existing meta-class (either *Class* or *Dependency*), we choose to represent them as stereotypes of these base meta-classes. Thus, we use for the meta-class *Class* the stereotypes «*contract*», «*assumption*», «*guarantee*», and «*observer*» (which already exists in OMEGA and is reused for representing *SafetyProperty*). For contract relations, the stereotypes of *Dependency* that correspond to the meta-model elements are «*contractImplementation*», «*contractUse*», «*contractRefinement*» and «*contractConformance*».

As explained before, from the semantic point of view, *Contracts* are not handled in the same way as usual classes/blocks as they are not considered executable elements of the system. Contracts are only used by the verification tools to check the validity of the conformance, dominance and satisfaction relations. The choice to describe contracts as *classes* is due to the fact that the syntax of contracts reuses much of the standard syntax of classes: we need to represent a contract as a closed composite structure with ports and links for the communication between the assumption and the guarantee and between the assumption and the component which uses the contract, interfaces to statically type the sets of inputs and outputs of each sub-component and behavior in the form of state machines.

## 4 The ATV Solar Generation System Case Study

The concepts and the reasoning method previously described have been applied on a case study, an industrial-grade system model of a subsystem of the Automated Transfer Vehicle (ATV). The ATV, developed by Astrium Space Transportation for the European Space Agency, is a spacecraft put into orbit by the European heavy launcher Ariane-5 with the aim of supplying the International Space Station. This case study consists of the Solar Wing Generation System (SGS) [9] responsible for the deployment and management of the solar wings of the vehicle. The SysML model

used in the following, provided by Astrium Space Transportation, was obtained by reverse engineering the actual SGS system for the purpose of this study.

The SGS system model[2] illustrated in Fig. 3 summarizes the three main components involved in the case study: the mission and vehicle management ($MVM$) part that initiates SGS wing deployment, the $SOFTWARE$ part of the $SGS$ that based on requests received from the $MVM$ executes the corresponding automated procedures and the $HARDWARE$ part that models the four physical wings. The communication between components is realized via asynchronous signals transported through ports and connectors. Due to the large number of ports (661) and connectors (504), Fig. 3 presents a simplified architectural view of SGS and only shows a link between two parts where several connectors and ports are involved in the actual model. Let us mention that the verification steps presented below have been performed on the initial system model.

Under the hypothesis that at most one hardware failure may occur during a run, which is embedded in the $HARDWARE$ model, the main goal of the case study is to verify the following property $\varphi$: after 10 minutes from system start-up, all four wings are deployed.

Due to the size and the complexity of the model, applying model-checking directly leads to combinatorial explosion and the verification of $\varphi$ does not finish. We explain in the following how the property $\varphi$ was verified using the contract-based reasoning methodology. We start by modeling the property $\varphi$. This implies identifying what the observer corresponding to the safety property $\varphi$ must monitor. In our case, the block *phi* must observe the answer that each wing provides with respect to its status (deployed or not deployed) when interrogated by the software. So, the property $\varphi$ expressed with respect to wing behavior must be satisfied by the $HARDWARE$ block instance that contains them. With regard to Fig. 1, the $HARDWARE$ is the subsystem $S$ and $WINGi$, $i = \overline{1,4}$, are the components $K_i$. The environment of the subsystem is given by the parts with which it communicates: bidirectional communication is directly established between $SOFTWARE$ and $HARDWARE$, while $SOFTWARE$ depends on the behavior of $MVM$. Thus, the environment $E$ of Fig. 1 is represented here by the composition of $MVM$ and $SOFTWARE$.

Next, we provide a contract $C = (A, G)$ such that it conforms to $\varphi$. In order to comply to the contract methodology, $C$ is implemented by $HARDWARE$'s type and it is used by this part within the proof tree. We use as assumption $A$ the concrete environment of $HARDWARE$, the composition between $MVM$ and $SOFTWARE$ itself, which thus satisfies by construction the mirror contract $C^{-1} = (G, A)$. Keeping this composition as assumption is not problematic since its state space has a manageable size. As guarantee $G$ we use the following abstraction derived (manually) from the individual behavior of wings: for each wing status interrogation, the target wing answers either as not deployed for at most 400 seconds or as deployed after at least 130 seconds. In order to ensure that the contract defines a closed system, since $MVM \| SOFTWARE$ sends all possible requests to $HARDWARE$, we equip $G$ with all ports defined by $HARDWARE$ and we enrich the behavior of $G$ to ignore all other requests. Then $G$ and $HARDWARE$ have the same set of ports: no refinement of requests is performed and the corresponding rule is satisfied.

The third step consists in modeling a set of contracts $\{C_1, C_2, C_3, C_4\}$ that refine $C$ and proving that each contract $C_i = (A_i, G_i)$ is implemented by $WINGi$'s type, $i = \overline{1,4}$. The environment for $WINGi$ is given by the environment of the subsystem $HARDWARE$ and all $WINGj$, $j \neq i$. We use the following abstraction $WAj$ for $WINGj$: the wing is either not deployed for at most 400 seconds or deployed from at least 130 seconds while all other received requests are consumed. The assumption $A_i$ is the parallel composition of $MVM$, $SOFTWARE$ and $WAj$, $j \neq i$. The guarantee $G_i$

[2] The case study is represented using the notation conventions of IBM Rhapsody: http://www-03.ibm.com/software/products/us/en/ratirhap/.

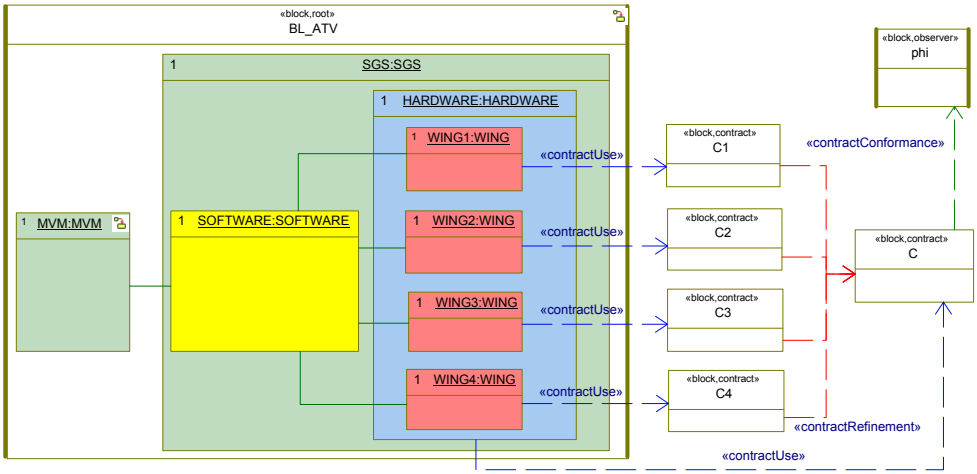


Figure 3: Architecture of the SGS system including contracts (simplified view).

is the projection of $G$ on *WINGi*. Again the refinement of requests is not considered and since the ports of *HARDWARE* with respect to *WINGi* are identical to the ports of *WINGi*, both rules on port set inclusion for implementation and refinement are satisfied. Moreover, each $C_i$ defines a closed system.

After this step the proof obligation tree is complete. The verification involves 10 intermediate steps: 4 for verifying that each wing satisfies its contract, 5 sufficient conditions for dominance between $\{C_1, C_2, C_3, C_4\}$ and $C$ and one for proving that $A \parallel G \preceq \varphi$. Each verification step is performed by the OMEGA-IFx model checker in a few hours. The overall effort of the building the contracts and performing the verification steps was of about 5 person*days. Even though the translation and verification algorithm are automated, two intermediate modeling steps - connecting assumptions to components via links and transforming the state machine of guarantees into observer (timed trace inclusion is verified using observers) - remain manual. The automation of these steps is currently under development. For further details on contract-based verification of the SGS case study, the reader is referred to [11].

## 5 Related Work

Modeling and verifying contracts for components is a long line of research, whose origins date back to Hoare logic [12]. Syntactical and behavioral contracts, as classified in [3], have been explored for specifying composition constraints and pre/post conditions for operations and also for modeling transformation of models and execution semantics. Contracts as a language construct have emerged with the Eiffel programming language [16] and have since been explored for various programming and specification models. In this section we concentrate on work aiming to introduce contracts in high-level modeling languages. For a discussion of more theoretical works on contracts and contract-based verification the reader is referred to [11].

*Weis et al.* [23] propose to model a contract for a component in UML by an interface and to specify its role: it can be either a *required* contract on which the component depends or a *provided* contract that is realized by the component. Syntactically, this representation of contracts is similar to ours: we also make the distinction between the required behavior of the environment and the provided behavior of the component by taking into account the assumption over the environment. However, our contracts are richer since they model a behavior that can be used for component validation,

while the contracts of [23] can be used only for composability checking during the development phases.

The Kmelia component model [15, 2], based on the work described above, provides means to verify the functional correctness of behavioral contracts for services: the behavior of an operation is modeled as a Labeled Transition System and formal verification can be realized within different tools via model transformation. Their meta-model defines for a contract the source implementing it as an aggregated element (operation or interface) and models explicitly the contract satisfaction results. But this formalism does not describe how the order in which services are called by and from a component can be verified, order that can be seen similar to our state machines from components. Furthermore, it does not provide a connection to high-level modeling languages as UML/SysML.

Contracts modeled as pre/post conditions are used in [7] for the verification of model transformation: the assumption is represented by an OCL constraint on the source model and the guarantee is an OCL constraint on the target model. Moreover, with respect to the syntax of contracts the two approaches are different: while the one we describe considers contracts only for components, [7] models contracts for all model elements. The same contracts are used in [6] to model the execution semantics of UML elements which is seen as a particular case of model transformation.

To the best of our knowledge, this study is the first to consider behavioral contracts at the component level in UML/SysML and to provide verification relations for property satisfaction by contract-based reasoning. The meta-model we propose is generic enough to represent all the other meta-models previously described, excepting the verification results extension that is based on the dynamical execution of the model.

## 6 Conclusion

Based on a theory of contracts and on a methodological approach for reasoning with contracts introduced in previous work [11], we have proposed an extension of UML/SysML allowing to model contracts and use them for compositional verification of requirements. The extension is defined as a meta-model, enriched with constraints and well-formedness rules to make contracts verifiable. We have instantiated the extension within the OMEGA UML/SysML profile to make it usable with standard model editors. The verification method is supported by the OMEGA-IFx toolset and the approach was validated on an industrial-grade system model.

Although an automatic model transformation from OMEGA system models to the input language of the IFx Toolset is already available, some of the steps for generating the intermediate contract-based verification models remain manual. Future work consists in automating all the intermediate model generation steps and in adding functionality for managing the proof obligations and results and for enforcing the rigorous verification methodology described in §2.3.

The method described before does not explicitly prescribe how to derive contracts for the whole system and for its components. In the case study described in §4, this task was relatively straightforward: since we make no additional assumption ($A$) about the environment, $G$ is roughly the same as $\varphi$, and the component guarantees are a projection of the desired global guarantee. There may be cases where the definition of contracts is less obvious and the overhead is significant, and previous attempts to introduce contracts in programming have not enjoyed a widespread success due to this kind of overhead. Nevertheless, we believe that the case for contracts in early phases of system engineering is different than the case for software programming, and the overhead should be acceptable for certain critical systems. Further work is needed in order to lower the overhead by finding methods or methodological guidelines for deriving intermediate contracts from the properties one is trying to prove.

## References


[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[2] P. André, A. Gilles, and M. Messabihi. Vérification de contrats logiciels a l'aide de transformations de modeles. In *7èmes journées sur l'Ingénierie Dirigée par les Modèles (IDM) 2011*, 2011.

[3] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making Components Contract Aware. *Computer*, 32(7):38–45, July 1999.

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

[5] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 237–267. Springer, 2004.

[6] E. Cariou, C. Ballagny, A. Feugas, and F. Barbier. Contracts for model execution verification. In *7th European conference on Modelling foundations and applications (ECMFA) 2011*, pages 3–18. Springer, 2011.

[7] E. Cariou, N. Belloir, F. Barbier, and N. Djemam. OCL contracts for the verification of model transformations. *ECEASST*, 24, 2009.

[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 *6th International Symposium on Embedded Real Time Software and Systems (ERTS2) 2012*. Online website, 2012.

[9] I. Dragomir, I. Ober, and D. Lesens. A case study in formal system engineering with SysML. In *17th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS) 2012*, pages 189–198. IEEE Computer Society, 2012.

[10] I. Dragomir, I. Ober, and C. Percebois. Integrating verifiable Assume/Guarantee contracts in UML/SysML. Technical report, IRIT, july 2013. Available at `http://www.irit.fr/~Iulian.Ober/docs/TR-Syntax.pdf`.

[11] I. Dragomir, I. Ober, and C. Percebois. Safety Contracts for Timed Reactive Components in SysML. Technical report, IRIT, june 2013. Submitted for publication. Available at `http://www.irit.fr/~Iulian.Ober/docs/TR-Contracts.pdf`.

[12] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, 1969.

[13] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata - Second Edition*. Morgan & Claypool Publishers, 2010.

[14] RTCA Inc. Software Considerations in Airborne Systems and Equipment Certification. Document RTCA/DO-178C, 2011.

[15] M. Messabihi, P. André, and C. Attiogbé. Multilevel Contracts for Trusted Components. In *International Workshop on Component and Service Interoperability*, volume 37 of *EPTCS*, pages 71–85, 2010.

[16] B. Meyer. Applying Design by Contract. *Computer*, 25(10):40–51, Oct. 1992.

[17] I. Ober and I. Dragomir. OMEGA2: A New Version of the Profile and the Tools. In *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS) 2010*, pages 373–378. IEEE Computer Society, 2010.

[18] Object Management Group. Systems Modelling Language (SysML) v1.1, 2008.

[19] Object Management Group. Unified Modelling Language (UML) v2.2, 2009.

[20] Object Management Group. Object Constraint Language (OCL) v2.2, 2010.

[21] S. Quinton. *Design, vérification et implémentation de systèmes à composants*. PhD thesis, Université de Grenoble, 2011.

[22] S. Quinton and S. Graf. Contract-based verification of hierarchical systems of components. In *Sixth IEEE International Conference on Software Engineering and Formal Methods (SEFM) 2008*, pages 377–381, 2008.

[23] T. Weis, C. Becker, K. Geihs, and N. Plouzeau. A UML Meta-model for Contract Aware Components. In *4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools (UML) 2001*, pages 442–456. Springer, 2001.