Safety Contracts for Timed Reactive Components in 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. A variety of system design and architecture description languages, such as SysML, UML or AADL, allows the decomposition of complex system designs into communicating timed components. In this paper we consider the contract-based specification of such components. A contract is a pair formed of an assumption, which is an abstraction of the component's environment, and a guarantee, which is an abstraction of the component's behavior given that the environment behaves according to the assumption. Thus, a contract concentrates on a specific aspect of the component's functionality and on a subset of its interface, which makes it relatively simpler to specify. Contracts may be used as an aid for hierarchical decomposition during design or for verification of properties of composites. This paper defines contracts for components formalized as a variant of timed input/output automata, introduces compositional results allowing to reason with contracts and shows how contracts can be used in a high-level modeling language (SysML) for specification and verification, based on an example extracted from a real-life system.

1 Motivation and Approach

The development of safety critical real-time embedded systems is a complex and costly process, and the early validation of design models is of paramount importance for satisfying qualification requirements, reducing overall costs and increasing quality. Design models are validated using a variety of techniques, including design reviews [24], simulation and model-checking [19, 25]. In all these activities system requirements play a central role; for this reason processes-oriented standards such as the DO-178C [22] emphasize the necessity to model requirements at various levels of abstraction and ensure their traceability from high-level down to detailed design and coding.

Since the vast majority of systems are designed with a component-based approach, the mapping of requirements is often difficult: a requirement is in general satisfied by the collaboration of a set of components and each component is involved in satisfying several requirements. A way to tackle this problem is to have partial and abstract component specifications which concentrate on specifying how a particular component collaborates in realizing a particular requirement; such a specification is called a *contract*. A contract is defined as a pair formed of an *assumption*, which is an abstraction of the component's environment, and a *guarantee*, which is an abstraction of the component's behavior given that the environment behaves according to the assumption.

The justification for using contracts is therefore manyfold: support for requirement specification and decomposition, mapping and tracing requirements to components and even for model reviews. Last but not least, contracts can support formal verification of properties through model-checking. Given the right composability properties, they can be used to restructure the verification of a property by splitting it in two steps: (1) verify that each component satisfies its contract and (2) verify that the network of contracts correctly assembles and satisfies the property. Thus, one only needs to reason on abstractions when verifying a property, which potentially induces an important reduction of the combinatorial explosion problem.

Our interest in contracts is driven by potential applications in system engineering using SysML [23], in particular in the verification of complex industrial-scale designs for which we have reached the limits of our tools [16]. In SysML one can describe various types of communicating timed reactive components; for most of these, their semantics can be given in a variant of Timed Input/Output Automata (TIOA) [21]. For this reason, in this paper we concentrate on defining a contract theory for TIOA. This contract theory is applied on a SysML case study extracted from real-life.

2 A Meta-Theory for Contract-based Reasoning

Our contract theory is an instance of the *meta-theory* proposed in [27] and later detailed in [26], which formalizes the relations that come into play in a contract theory and the properties that these relations have to satisfy in order to support reasoning with contracts. The term *meta-theory* refers to the fact that the formalism used for component specification is not fixed, nor the exact nature of certain relations defined on specifications (conformance, refinement under context). In order to obtain a concrete contract theory for a particular specification formalism one has to define these relations such that certain properties, prerequired by the meta-theory, are satisfied. In return, this meta-theory provides a ready-to-use contract-based methodology.

The purpose of this methodology is to support reasoning with contracts in a system obtained by hierarchical composition of components. At any level of the hierarchy, n components $K_1, ..., K_n$ are combined to form a composite component $K_1 \parallel ... \parallel K_n$, where $\parallel$ denotes the parallel composition operator. Then verifying that the composite satisfies a global property φ runs down to checking that the contracts implemented by $K_1, ..., K_n$ combine together correctly to ensure φ. This avoids the need to directly model-check the composite to establish φ and, so, alleviates the combinatorial explosion of the state space. The contracts being specified by more abstract automata, one can assume that their composition will be reduced.

The reasoning proceeds as follows: for each component K_i, a contract C_i is given which consists of an abstraction A_i of the behavior of K_i's environment, and an abstraction G_i that describes the expected behavior of K_i given that the environment acts as A_i. Fig. 1 presents three components K_1, K_2 and K_3 and a corresponding set of contracts C_1, respectively C_2 and C_3. Step 1 of the reasoning consists in verifying that each component is a correct implementation of the contract, i.e. the component *satisfies* its contract. The *satisfaction* relation is directly derived from a more general one named *refinement under context*. The purpose of the latter is to model that a component K_i is a correct refinement of the specification K_j in the given environment E_k. Thus, a component implements

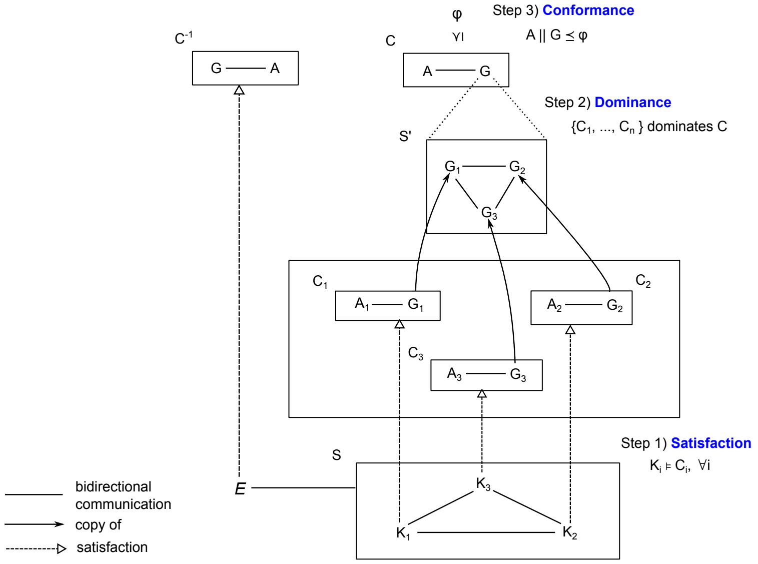

Fig. 1: Contract-based reasoning for a three-component subsystem ([26]).

a contract if and only if *refinement under context* holds between the component K_i and the guarantee G_i in the environment A_i.

Step 2 of the reasoning consists in defining a contract $C = (A, G)$ for the composition $K_1 \parallel ... \parallel K_n$ and proving that the set of contracts $\{C_1, C_2, ..., C_n\}$ *implies* C. To do so, the meta-theory introduces a hierarchy relation between contracts, called *dominance*: a set of contracts $\{C_1, C_2, ..., C_n\}$ *dominates* a contract C if and only if the composition of any valid implementations of $C_1, C_2, ..., C_n$ is an implementation of C. In a multi-level hierarchy, the second step can be applied recursively up to a top-level contract, i.e. a contract for the whole (sub)system.

Finally, in the third step one has to prove that the top contract $C = (A, G)$ implies the specification φ. This is done by verifying that $A \parallel G \preceq \varphi$, where $\preceq$ is a *conformance* relation. This step is sufficient for proving that the whole system satisfies φ if and only if either the assumption A is empty as it is the case when the property is defined for the entire closed modeled system or A is a correct abstraction of the environment E with which S communicates given that S behaves like G, i.e. E *satisfies* the "mirror" contract $C^{-1} = (G, A)$.

The reasoning strategy presented here assumes that the system designer defines all the contracts necessary for verifying a particular requirement φ. How these contracts are produced is an interesting question but it is outside the scope of this paper.

The theoretical contribution of this paper is the instantiation of this meta-theory for a variant of Timed Input/Output Automata [21], by choosing the appropriate refinement relations and proving that they satisfy certain properties needed for the meta-theory to be applied. Concretely, the difficulties of the theoretical work consisted in defining the *conformance* and *refinement under context* relations such that one can reason in a contract framework only by handling assumptions and guarantees once satisfaction has been proven. The refinement relation has to satisfy the *compositionality* property and the soundness of *circular reasoning*, detailed in §4. *Compositionality* allows for *incremental design* by incorporating parts of the environment into the component under study while refinement under context holds at every step. *Circular reasoning* allows for *independent implementability* by breaking the dependency between the component and the environment.

The practical contribution of this paper is the application of the contract framework to a case study modeled in SysML, which can be found in §5. Due to space limitations we skip the syntactic details of how contracts are expressed in SysML and we concentrate on describing the example, the property of interest and the contracts involved in proving it. The relatively complex SysML language aspects are detailed in [17].

3 Timed Input/Output Automata

Many mathematical formalisms have been proposed in the literature for modeling communicating timed reactive components. We choose to build our framework on a variant of Timed Input/Output Automata of [21] since it is one of the most general formalisms, thoroughly defined and for which several interesting compositional results are already available.

A *timed input/output automaton* (or *component*) defines a set of internal variables of arbitrary types and clocks and a set of *input* actions I, *output* actions O, *visible* actions V and *internal* actions H. We denote by $E = I \cup O \cup V$ the set of external actions that also gives the *interface* of the component and by $A = E \cup H$ the set of actions. The state space of an automaton is described by the set of possible valuations of the set of variables. The state evolves either by discrete transitions or by *trajectories*. A discrete transition instantly changes the state and is labeled with an action from the mentioned sets. Trajectories change the state continuously during a time interval.

The behavior of a TIOA is described by an *execution fragment* which is a finite or infinite sequence alternating trajectories and discrete transitions. The *visible* behavior of a TIOA is given by a *trace*, which is a projection of an execution fragment on external actions and in which, from trajectories, only the information about the elapsed time is kept, while information about the variable valuations is abstracted away. For full definitions of all these notions, the reader is referred to the long version of this paper [18] and to [21].

The difference between the TIOA of [21] and our variant is that in addition to *inputs* and *outputs*, we allow for another type of *visible* actions; this is because, in [21], when composing two automata, an *output* of one matched by an *input* of the other becomes an *output* of the composite, which does not correspond to our needs when using TIOA for defining the semantics of usual modeling languages like SysML. Indeed, the matching between an input and an output results in a visible action in SysML that is not further involved in any synchronizations.

Moreover, in the following we will limit our attention to trajectories that are constant functions for discrete variables, and linear functions with derivative equal to 1 for clocks, while [21] allows more general functions to be used as trajectories. This restriction makes the model expressiveness equivalent to that of Alur-Dill timed automata [1], and will be important later on as it opens the possibility to automatically verify simulation relations between automata (simulation is undecidable for the TIOA of [21]). However, this hypothesis is not needed for proving the compositional results in §4.

The parallel composition operator for TIOA is based on binary synchronization of corresponding inputs and outputs and on the interleaved execution of other actions, like in [21]. The only difference is related to the interface of the composite timed input/output automata: the input and output sets of the composite are given by those actions not matched between components, while all matched input-output pairs become visible actions. Two automata can be composed if and only if they do not share variables and internal and visible actions. Two automata that can be composed are called *compatible components*. As in [21], we use timed trace inclusion as the refinement relation between components.

Definition 1 (Comparable components). *Two components K_1 and K_2 are* comparable *if they have the same external interface, i.e. $E_{K_1} = E_{K_2}$.*

Definition 2 (Conformance). *Let K_1 and K_2 be two comparable components. K_1* conforms to *K_2, denoted $K_1 \preceq K_2$, if $traces_{K_1} \subseteq traces_{K_2}$.*

The conformance relation is used in the third step for verifying the satisfaction of the system's properties by the top contract: $A \parallel G \preceq \varphi$, where $A \parallel G$ and φ have the same interface. It can be easily shown that conformance is a preorder. The following useful compositional result (*theorem 8.5* of [21]) can be easily extended to our variant of TIOA:

Theorem 1. *Let K_1 and K_2 be two comparable components with $K_1 \preceq K_2$ and E a component compatible with both K_1 and K_2. Then $K_1 \parallel E \preceq K_2 \parallel E$.*

4 Contracts for Timed Input/Output Automata

In this section we formalize contracts for TIOA and the relations described in §2 and we list the properties that we proved upon these and that make contract-based reasoning possible.

Definition 3 (Environment). *Let K be a component. An* environment *E for the component K is a timed input/output automaton compatible with K for which the following hold: $I_E \subseteq O_K$ and $O_E \subseteq I_K$.*

Definition 4 (Closed/open component). *A component K is* closed *if $I_K = O_K = \emptyset$. A component is* open *if it not closed.*

Closed components result from the composition of components with complementary interfaces.

Definition 5 (Contract). *A* contract *for a component K is a pair (A, G) of TIOA such that $I_A = O_G$ and $I_G = O_A$ (i.e. the composition pair $A \parallel G$ defines a closed component) and $I_G \subseteq I_K$ and $O_G \subseteq O_K$ (i.e. the interface of G is a subset of that of K). A is called the* assumption *over the environment of the component and G is called the* guarantee. *The* interface of a contract *is that of its guarantee.*

The first step of the verification, as presented in §2, is to prove that the modeled components are an implementation of the given contracts. For this, we firstly define the *refinement under context* preorder relation which, in our framework, is further based on conformance. The complete formal definition of *refinement under context* can be found in [18].

Definition 6 (Refinement under context). *Let K_1 and K_2 be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let E be an environment for K_1 compatible with both K_1 and K_2. We say that K_1* refines *K_2* in the context of *E, denoted $K_1 \sqsubseteq_E K_2$, if*

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where

- *E' is a TIOA defined such that the composition $K_1 \parallel E \parallel E'$ is closed. E' consumes all outputs of K_1 not matched by E and may emit all inputs of K_1 not appearing as outputs of E.*
- *K' is a TIOA defined similarly to E' such that the composition of $K_2 \parallel E \parallel K' \parallel E'$ is closed and comparable to $K_1 \parallel E \parallel E'$.*

Since we want to take into account interface refinement between components and conformance imposes comparability, we have to compose each member of the conformance relation obtained from refinement under context with an additional timed input/output automaton E', respectively K', such that they both define closed comparable systems. Both automata are uniquely defined by their interfaces and can be automatically computed.

Furthermore, the particular inclusion relations between the interfaces of K_1 and K_2 in the previous definition are due to the fact that both K_1 and K_2 can be components obtained from compositions, e.g., $K_1 = K_1' \parallel K_3$ and $K_2 = K_2' \parallel K_3$, where $I_{K_2'} \subseteq I_{K_1'}$, $O_{K_2'} \subseteq O_{K_1'}$ and $V_{K_2'} \subseteq V_{K_1'}$. This happens in particular when K_2' is a contract guarantee for K_1'. Then, by composition, actions of K_3 may be matched by actions of K_1' but have no input/output correspondent in K_2'.

Theorem 2. *Given a set $\mathcal{K}$ of comparable components and a fixed environment E for their interface, refinement under context $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.*

We derive the *satisfaction* relation from refinement under context.

Definition 7 (Contract satisfaction). *A component K* satisfies (implements) *a contract $C = (A, G)$, denoted $K \models C$, if and only if $K \sqsubseteq_A G$.*

We have introduced the notions and relations in order to verify that a component is a correct implementation of a contract. The second step of the contract-based verification methodology relies on the notion of dominance, introduced informally in §2, which is formally defined in [26] as follows:

Definition 8 (Contract dominance). *Let C be a contract with the interface $\mathcal{P}$ and $\{C_i\}_{i=1}^n$ a set of contracts with the interface $\{\mathcal{P}_i\}_{i=1}^n$ and $\mathcal{P} \subseteq \bigcup_{i=1}^n \mathcal{P}_i$. Then $\{C_i\}_{i=1}^n$* dominates *C if and only if for any set of components $\{K_i\}_{i=1}^n$ such that $\forall i,\ K_i \models C_i$, we have $(K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C$.*

In order to ease dominance verification by discarding components from now on and to be able to apply the meta-theory, we prove that the following compositional results hold in our framework.

Theorem 3 (Compositionality). *Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$. Then $K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \Leftrightarrow K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$.*

Theorem 4 (Soundness of circular reasoning). *Let K be a component, E its environment and $C = (A, G)$ the contract for K such that K and G are compatible with each of E and A. If (1) $traces_G$ is closed under limits, (2) $traces_G$ is closed under time-extension, (3) $K \sqsubseteq_A G$ and (4) $E \sqsubseteq_G A$ then $K \sqsubseteq_E G$.*

The definitions of closure under limits and closure under time-extension for a set of traces are those given in [21]. Closure under limits informally means that any infinite sequence whose finite prefixes are traces of G is also a trace of G, while closure under time-extension denotes that any finite trace can be extended with time passage to infinity. By making these hypotheses on G, G can only express safety properties on K and cannot impose stronger constraints on time passage than K. The proofs of all theorems presented here can be found in [18].

Then based on theorems 2, 3 and 4, the following theorem also proved in [18] which is a variant of *theorem 2.3.5* from [26] holds:

Theorem 5 (Sufficient condition for dominance). *$\{C_i\}_{i=1}^n$ dominates C if, $\forall i$, $traces_{A_i}$, $traces_{G_i}$, $traces_A$ and $trace_G$ are closed under limits and under time-extension and*

$$\begin{cases} G_1 \parallel ... \parallel G_n \sqsubseteq_A G \\ A \parallel G_1 \parallel ... \parallel G_{i-1} \parallel G_{i+1} \parallel ... \parallel G_n \sqsubseteq_{G_i} A_i, \ \forall i \end{cases}$$

The above theorem specifies the proof obligations that have to be satisfied by a system of contracts in order to be able to infer dominance in the second step of the verification methodology presented in §2.

5 Application to a SysML Model: the ATV Solar Wing Generation System Case Study

The contract-based reasoning method previously described is partially supported by the OMEGA-IFx Toolset [7] for SysML models. The details of the SysML language extended with contracts are left aside for space reasons and can be found in [17]. In the following we present a case study extracted from the industrial-grade system model of the Automated Transfer Vehicle (ATV) and we show how contracts can be used for property verification. This case study consists of the Solar Wing Generation System (SGS) [16] 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 ATV system model illustrated in Fig. 2 summarizes the three main components involved in the case study and the bidirectional communications between them: the mission and vehicle management (MVM) part that initiates the two functionalities of the SGS (wing deployment and rotation), the $SOFTWARE$ part of the SGS that based on commands received from the MVM executes the corresponding procedures and the $HARDWARE$ part that consists of the four wings. We focus here on the wing deployment mode on which we want to verify the following property φ: after 10 minutes from system start-up, all four wings are deployed.

The system explicitly models the redundancy of the hardware equipments which aims to ensure fault tolerance. There are 56 possible failures (14 per wing)

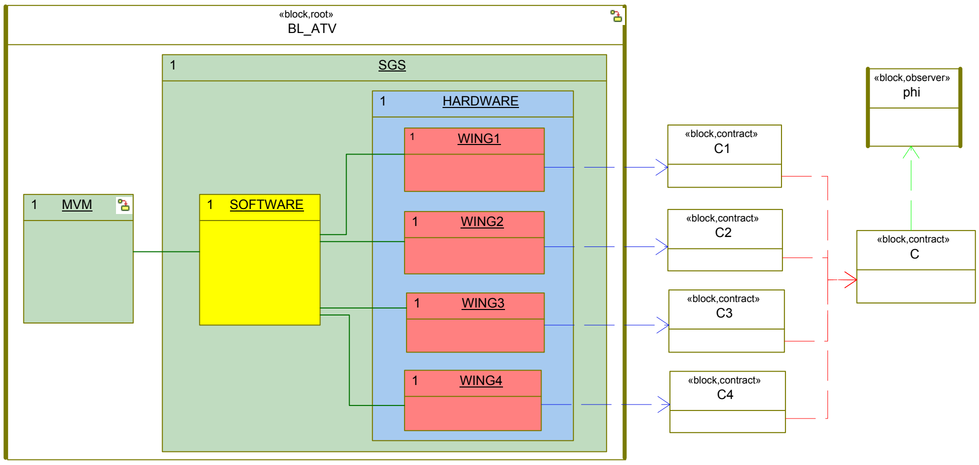

Fig. 2: Architecture of the SGS system including contracts (simplified view).

grouped in 3 classes depending on their target (thermal knives, hold-down and release system and solar array driving group). The following hypothesis is made: throughout any execution of the system, at most one hardware fault may occur (1-fault tolerance). We are interested in verifying φ by taking into consideration this hypothesis. But applying model-checking directly on the system leads to combinatorial explosion and the verification does not finish. To give an idea about the complexity of the model at runtime, the system contains 96 objects that communicate through 661 port instances and 504 connectors. In [16] we have shown the motivation for using contracts and we have sketched a proof that remained to be formalized and verified. In the following we complete this case study by proving the property φ with formalized contract-based reasoning.

Since the property φ is expressed with respect to the behavior of the four wings that are contained in the *HARDWARE* block, with regard to the methodology of Fig. 1, the subsystem S can be identified in our case study with *HARDWARE* and the components K_i are represented by *WINGi*, $i = \overline{1,4}$. 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*. So, the environment E of Fig. 1 is represented here by the composition of *MVM* and *SOFTWARE*.

The first step of the methodology consists in defining a contract $C_i = (A_i, G_i)$ for each *WINGi*, and next proving that *WINGi* satisfies C_i, $i = \overline{1,4}$. This step checks the validity of the dependency relations between the wings and their corresponding contracts. In order to model a contract, first we need to identify the environment of the component to which the contract is associated and to build an abstraction from the point of view of the component. Thus, for *WINGi* the environment is given by the environment of the subsystem *HARDWARE* and all *WINGj* with $j \neq i$. We propose the following abstraction *WAj* for *WINGj*: the wing has a not deployed status for at most 400 seconds and a deployed status after 130 seconds, while all other received requests are consumed. The assumption A_i is then given by the parallel composition of *MVM*, *SOFTWARE* and

WAj with $j \neq i$. This abstraction of the environment is sufficient to drastically reduce the state space of the verification model, since the exponential explosion in the original model is mainly due to the parallelism of the hardware pieces which are abstracted to the three leaf parts *WAj*. We want to guarantee that even if *WINGi* exhibits a failure it ends up being deployed after 400 seconds.

Contract $C_i = (A_i, G_i)$ where

- $A_i = MVM \parallel SOFTWARE \parallel (\parallel_{j \neq i} WAj)$.
- G_i: the wing answers to requests about its status with *not deployed* from startup up to 400 seconds or with *deployed* after 130 seconds and ignores all other requests. Between 130 and 400 seconds it can answer either, nondeterministically.

Since A_i is partially given by the concrete environment ($MVM \parallel SOFTWARE$) and C_i has to define a closed system, we have to manually model the behavior of G_i for all received requests. This constraint imposes to add as consuming transitions in every state all requests corresponding to wing deployment process. Furthermore, one can remark that this guarantee is stronger than the projection of the property φ on *WINGi*. The abstraction *WAj* can also be subject to one failure since this case was not excluded from its behavior. Then the fault tolerance property that we verify via contracts is stronger than the initial hypothesis: we guarantee that the system is 4-fault tolerant if faults occur in separate wings.

The second step consists in defining a global contract $C = (A, G)$ for *HARDWARE* and to prove that the contract is dominated by $\{C_1, C_2, C_3, C_4\}$ represented in Fig. 2 by the dependency relation between contracts. We use as assumption A the concrete environment of *HARDWARE*. The guarantee G is the composition of the four G_i. All A_i, G_i, A and G as defined satisfy the conditions for applying theorem 5.

Contract $C = (A, G)$ where

- $A = MVM \parallel SOFTWARE$
- G : for each wing status interrogation it answers as *not deployed* for at most 400 seconds and as *deployed* after at least 130 seconds, while all other requests are ignored.

The last step consists in verifying that the composition $A \parallel G$ conforms to φ, illustrated by the dependency between the contract and the property. Verifying that the environment satisfies the "mirror" contract is trivial since the assumption A is the environment itself.

The proofs of all three steps have been automatically verified within the OMEGA-IFx Toolset which translates SysML models in communicating timed automata [7]. Since trace inclusion is undecidable, we use a stronger *simulation* relation whose satisfaction implies trace inclusion. So, verifying simulation is sufficient (albeit not necessary) in order to prove the satisfaction of the conformance relation. A variant of this verification algorithm is implemented in the IFx Toolset.

For each step of the verification methodology we have manually modeled the contracts: assumptions as blocks that we had to connect via ports with the other components and guarantees as independent components. The first step gave 4 possible configurations with one concrete wing and 3 abstract ones that were each verified with respect to all 14 possible failures. The average time in seconds needed for the verification of the satisfaction relation for each contract

Table 1: Average verification time for each contract C_i per induced failure group.

	Average verification time (s)			
Type of induced failure	Wing 1	Wing 2	Wing 3	Wing 4
Thermal knife	13993	6869	18842	11412
Hold-down and release system	12672	6516	16578	9980
Solar array driving group	11527	5432	13548	6807

with respect to each class of failures is presented in Table 1. Even though the system model looks symmetrical, some hardware pieces not represented here do not have a symmetrical behavior and due to their interconnections with the wings the state space of system's abstraction for $WING1$ and $WING3$ is larger than the one of $WING2$ and $WING4$. For the second step, only one model is created on which we verified all 5 proof obligations given by theorem 5: the automatic validation of the global guarantee G and the automatic validation of assumptions A_i. Modeling the assumptions A_i that play the role of guarantees for dominance verification shows the symmetry of the MVM and $SOFTWARE$ behavior. This means that only one verification is in fact sufficient for proving all 4 relations, verification that was realized in 9 seconds. The verification of the guarantee G needed 1 second. Finally, the same model was used for verifying φ that took 1 second.

6 Related Work

Contract-based specification and reasoning for communicating components has been subject to intensive research recently. As mentioned in the beginning, our contract theory for TIOA is an instance of the meta-theory of [26], which has previously been applied for a number of other components formalisms: Labeled Transition Systems (with priorities) [26], Modal Transition Systems [27], BIP framework [2, 26] and Heterogeneous Rich Components [5]. To the best of our knowledge, this is the first documented application of this meta-theory to Timed Input/Output Automata.

Contract meta-theories have also been built on specification theories. The aim of a specification theory is to provide a complete algebra with powerful operators for logical compositions of specifications and for synthesis of missing components (quotient), the main goal being to provide substitutability results allowing for compositional verification. The meta-theory of [3] falls under this category. The main differences with respect to [26] concern (1) the definition of contracts that do not support signature refinement (a partial solution for this problem has been proposed in [4]) and (2) the method for reasoning with contracts which relies on a *contract composition* operator that is partially defined. However, the meta-theory of [4] does not alleviate this problem. None of these two meta-theories supports circular arguments nor defines a methodology for reasoning with contracts. Moreover, the meta-theories do not provide means to formalize requirements and to verify their satisfaction. Other approaches for reasoning with contracts have been developed for interface theories [11, 12] and for logical specifications [10]. A complete comparison between (meta-)theories is available in [18].

Only the meta-theory of [3] has been instantiated for a variant of TIOA in [14, 15, 12] which is implemented in the ECDAR tool [13]. However, several aspects of this specification theory make it unsuitable for representing the semantics of timed components described in SysML or UML. The synchronization between an input of one component and an output of another component becomes an output of the composite, which equates to considering outputs as broadcasts and which is not consistent with the UML/SysML semantics. Moreover, the formalism forbids non-determinism due to the timed game semantics [6] and does not handle *silent* transitions, which is problematic for representing the semantics of complex components performing internal computation steps.

In addition, contracts in UML/SysML have until now been explored for the specification of composition compatibility of components via interfaces [28] and for the verification of pre/post conditions of operations as presented by [20]. Recent work covers the use of pre/post condition contracts for modeling transformation of models [9] and for modeling the execution semantics of UML elements [8]. To the best of our knowledge, our work is the first on using assume/guarantee behavioral contracts for the verification of UML/SysML model requirements.

7 Conclusions

We have presented a contract framework for Timed Input/Output Automata and results which allow contract-based reasoning for verifying timed safety properties of systems of TIOA components. We have illustrated the method on a case study extracted from an industrial-scale system model and we have showed how contract-based reasoning can alleviate the problem of combinatorial explosion for the verification of large systems.

The present work is a step further towards introducing contracts in SysML and providing a full solution to that problem. In [17] we defined a suitable syntax for contracts in SysML and a set of well-formedness rules that system models must satisfy for reasoning with contracts. For the moment, some steps of the method applied on SysML remain manual like modeling individual systems for each contract satisfaction relation or for each dominance proof obligation. Future work includes: (1) formalizing the semantic mapping between SysML components and contracts and their TIOA counterparts and (2) providing means for automatic verification by automated generation of proof obligations.

References

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[2] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *SEFM'06*, pages 3 –12, 2006.

[3] S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from Specifications to Contracts in Component-Based Design. In *FASE'12*, volume 7212 of *LNCS*, pages 43–58. Springer, 2012.

[4] S. Bauer, R. Hennicker, and A. Legay. Component Interfaces with Contracts on Ports. In *FACS'12*, volume 7684 of *LNCS*, pages 19–35. Springer, 2013.

[5] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis. A contract-based formalism for the specification of heterogeneous systems. In *FDL'08. Forum on*, pages 142–147. IEEE, 2008.

[6] T. Bourke, A. David, K. Larsen, A. Legay, D. Lime, U. Nyman, and A. Wasowski. New Results on Timed Specifications. In *WADT'10*, volume 7137 of *LNCS*, pages 175–192. Springer, 2010.

[7] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In *SFM'04*, volume 3185 of *Lecture Notes in Computer Science*, pages 237–267. Springer Berlin / Heidelberg, 2004.

[8] E. Cariou, C. Ballagny, A. Feugas, and F. Barbier. Contracts for model execution verification. In *ECMFA'11*, pages 3–18, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[10] C. Chilton, B. Jonsson, and M. Kwiatkowska. Assume-Guarantee Reasoning for Safe Component Behaviours. In *FACS'12*, volume 7684 of *LNCS*, pages 92–109. Springer, 2013.

[11] C. Chilton, M. Kwiatkowska, and X. Wang. Revisiting Timed Specification Theories: A Linear-Time Perspective. In *FORMATS'12*, volume 7595 of *LNCS*, pages 75–90. Springer, 2012.

[12] A. David, K. G. Guldstrand Larsen, A. Legay, M. H. Møller, U. Nyman, A. P. Ravn, A. Skou, and A. Wasowski. Compositional verification of real-time systems using ECDAR. *STTT*, 14(6):703–720, 2012.

[13] A. David, K. Larsen, A. Legay, U. Nyman, and A. Wasowski. ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems. In *ATVA'10*, volume 6252 of *LNCS*, pages 365–370. Springer, 2010.

[14] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Methodologies for Specification of Real-Time Systems Using Timed I/O Automata. In *FMCO'09*, volume 6286 of *LNCS*, pages 290–310. Springer, 2010.

[15] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *HSCC'10*, pages 91–100. ACM, 2010.

[16] I. Dragomir, I. Ober, and D. Lesens. A case study in formal system engineering with SysML. In *ICECCS'12*, pages 189–198. IEEE Computer Society, 2012.

[17] I. Dragomir, I. Ober, and C. Percebois. Integrating verifiable Assume/Guarantee contracts in UML/SysML. In *ACES-MB'13*. CEUR Workshop Proceedings, 2013.

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

[19] E. A. Emerson and E. M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In *ICALP'80*, volume 85 of *LNCS*, pages 169–181. Springer, 1980.

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

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

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

[23] OMG. Object Management Group – Systems Modeling Language (SysML), v1.1. Available at http://www.omg.org/spec/SysML/1.1/, 2008.

[24] D. Parnas and D. Weiss. Active Design Reviews: Principles and Practices. In *ICSE'85*. IEEE Computer Society, 1985.

[25] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982.

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

[27] S. Quinton and S. Graf. Contract-Based Verification of Hierarchical Systems of Components. *SEFM'08*, pages 377–381, 2008.

[28] T. Weis, C. Becker, K. Geihs, and N. Plouzeau. A UML Meta-model for Contract Aware Components. In *UML'01*, pages 442–456. Springer-Verlag, 2001.