Safety contracts for timed reactive components (extended abstract*)

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, rely on 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 behaviour 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 and introduces compositional results allowing to reason with contracts

1 Introduction

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 [10], simulation and model-checking [4,11]. In all these activities system requirements play a central role; for this reason processes-oriented standards such as the DO-178C [7] 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 behaviour given that the environment behaves according to the assumption.

The justification for using contracts is therefore manyfold: they support requirement specification and decomposition, mapping and tracing requirements to components and can be used in model reviews. Last but not least, contracts can support formal verification of properties through model-checking since, 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 the components satisfy their corresponding contract and (2) 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.

Contracts have been introduced in object-oriented programming languages [8] and related concepts have since been defined for other component-based formalisms. Our contract framework is an instance of the generic framework proposed in [13,12], which formalizes the relations that come into play in such a framework and the properties that these relations have to satisfy in order to support reasoning with contracts.

Our interest in contracts is driven by potential applications in system engineering using SysML [9], in particular in the verification of complex industrial-scale designs for which we have reached the limits of our tools [3]. 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 [5]). For this reason, this work concentrates on defining a contract framework for TIOA. The SysML layer, describing how contracts are defined and used in SysML, is left aside for space and complexity reasons and is subject to future work.

2 Contract-based Reasoning for Timed Reactive Systems

Being able to specify a contract for a component and to verify its satisfaction is an important step, but it is not sufficient to render the use of contracts interesting in a system design process. One also needs mechanisms for reasoning with contracts, i.e. check that the contracts for a set of components combine together to ensure the satisfaction of a global requirement on the composition of these components. Our framework follows a generic scheme for reasoning with contracts proposed by Quinton et. al. in [13,12].

In this scheme (illustrated in Figure 1), a system design is a hierarchical composition of components, these being Timed Input/Output Automata in our case. At each level of the hierarchy, n components $K_1, ..., K_n$ are combined to form a composite component $K_1 \parallel ... \parallel K_n$. The purpose of reasoning with contracts is to show that the composite satisfies a global property φ based on the contracts of $K_1, ..., K_n$, and avoiding the need to verify the property directly by model-checking the composite component, since this often leads to combinatorial

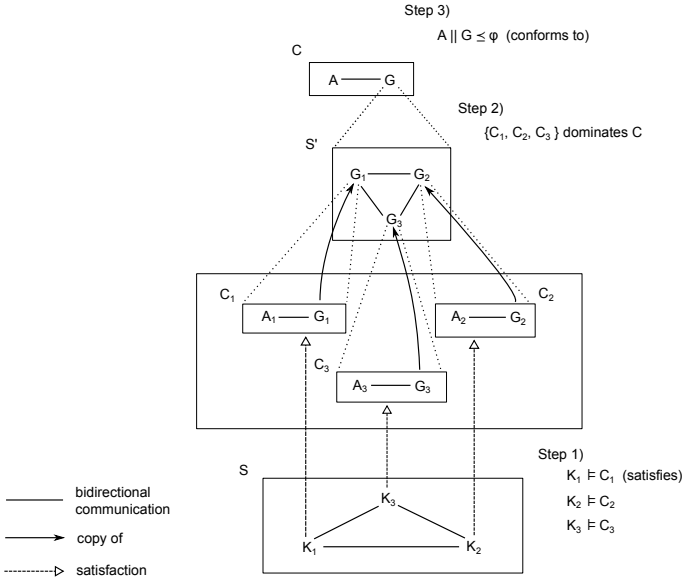

Fig. 1: Contract-based reasoning for a subsystem containing three components as presented in [12].

explosion. The contracts being specified by more abstract automata, one can assume that their composition will be less subject to explosion.

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 behaviour of K_i's environment, and an abstraction G_i that describes the expected behaviour of K_i given that the environment acts as A_i. Figure 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 is to verify that each component is a correct implementation of the contract, i.e. the component *satisfies* its contract.

Step 2 of the reasoning consists in proving that the set of contracts $\{C_1, C_2, ..., C_n\}$ *imply* that the composite $K_1 \parallel ... \parallel K_n$ satisfies a contract $C = (A, G)$. To do so, [6] introduces a hierarchy relation between contracts, later called *dominance* in [12]: 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 $C_1, C_2, ..., C_n$ (hence, also $K_1 \parallel ... \parallel K_n$) is an implementation of C. As we will see later, to prove dominance one will have to verify certain conditions on compositions of assumptions and guarantees. 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 system).

Finally, in the third step one has to prove that the system given by the top contract *conforms* to the specification φ (i.e. the property is satisfied): $A \parallel G \preceq \varphi$, where $\parallel$ denotes the usual parallel composition operator and $\preceq$ a *conformance* relation which in our case is TIOA language inclusion, φ also being specified as a TIOA.

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 is outside the scope of this paper.

3 Timed Input/Output Automata

Many mathematical formalisms have been proposed in the literature for modelling communicating timed reactive components. We chose to build our framework based on a variant of Timed Input/Output Automata of [5] since it is one of the most general formalisms, thoroughly defined and for which several interesting compositionality results are already available.

The state space of a TIOA is defined as a set of possible valuations of a set of variables with arbitrary types. The state evolves either by discrete transitions or by *trajectories*. A discrete transition instantly changes the state (i.e. variable valuations) and is labelled with an action that may be internal, an *input* or an *outputs*. Trajectories change the state continuously during a time interval. The behaviour of a TIOA is described by an *execution fragment* which is a finite or infinite sequence alternating trajectories and discrete transitions. The *visible* behaviour of a TIOA is described by a *trace*, which is a projection of an execution trace onto visible actions (inputs and outputs) and in which, from trajectories, only the information about the elapsed time is kept, and information about the variable valuations is abstracted away. For full definitions of all these notions, the reader is referred to [5].

There are two main differences between the TIOA of [5] and our variant:

- [5] allows general functions to be used as trajectories. We restrict ourselves to the identity function for clocks, and to the constant functions for discrete variables. 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 [5]). It also simplifies the presentation of examples, since trajectories are then fully determined by their domain, so we simply use the interval J to represent the trajectory. However, this hypothesis is not needed for proving the compositionality results in section 4.
- In addition to *inputs* and *outputs*, we allow for another type of *visible* actions; this is because, in [5], when composing two automata, an *output* of one matched by an *input* of the other become an *output* of the composite, which does not correspond to our needs when using the TIOA for defining the semantics of usual modelling languages like SysML. As we will show, in order for contract dominance to work, we still need the resulting action to be *visible* in traces, hence the necessity for an additional type of actions.

In the following, for a TIOA A, we denote I_A its set of inputs, O_A its outputs, V_A its visible actions, H_A its internal actions, $E_A = I_A \cup O_A \cup V_A$ and

$A_A = E_A \cup H_A$. The parallel composition operator for TIOA, defined similarly to [5], is denoted by $\|$. We sometimes use the term *component* instead of TIOA, interchangeably.

As in [5], we will use 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 definition of refinement under context and 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 compositionality result, presented in the Timed Input/Output Automata theory of [5], can be easily extended to our variant of TIOA:

Theorem 1 (Composability theorem 8.5 of [5]). *Let I and S be two comparable components with $I \preceq S$ and E a component compatible with both I and S. Then $I \parallel E \preceq S \parallel E$.*

4 Contracts for Timed Input/Output Automata

In this section we provide the definitions for TIOA contracts and the relations described in Section 2, and we list the properties that have been 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 for which the following hold: $I_E \subseteq O_K$ and $O_E \subseteq I_K$.*

Definition 4 (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 system) 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.

Definition 5 (Closed/open component). *A component K is closed if $I_K = O_K = \emptyset$. A component is open if it is not closed.*

In the following, closed components result from the composition open components with complementary interfaces.

The *refinement under context* relation verifies that, given an environment compatible with two components, one component is a refinement of the other in the specified environment. We define this relation with respect to conformance. Since we want to take into account interface refinement between the components

and conformance imposes comparability, we have to compose each member of the conformance relation with an additional timed input/output automaton such that they both define closed comparable systems.

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 $K_2 \parallel E \parallel K' \parallel E'$ is closed and comparable to $K_1 \parallel E \parallel E'$.*

The complete formal definition of E' and K' appears in the extended version of this paper.

The particular relationship required between the interfaces of K_1 and K_2 in the above definition is due to the fact that both K_1 and K_2 can be components obtained from composition, 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 action of K_1' but have no input/output correspondent in K_2'. This case also imposes the term $V_{K_1} \cap O_{K_2}$ for the inputs of K', since the additional outputs of K_2 may belong to a different component, and the term $V_{K_1} \cap I_{K_2}$ for the outputs of K'.

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

The following, required to allow reasoning with contracts, as shown in [12], hold in our framework:

Theorem 3 (Compositionality of refinement under context). *Let K_1 and K_2 be two components and E an environment compatible with both K_1 and K_2 such that $E = E_1 \parallel E_2$. If $K_1 \sqsubseteq_{E_1 \parallel E_2} K_2$ then $K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$.*

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 $traces_G$ is* closed under limits *and* closed under time-extension*, $K \sqsubseteq_A G$ and $E \sqsubseteq_G A$ 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 [5]. Closure under time extension informally means that any 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.

We define *contract satisfaction* based on 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 $K \sqsubseteq_A G$.*

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 that $(K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C$*

Based on theorems 2, 3 and 4, the following theorem, which is a variant of Theorem 2.3.5 from [12] holds:

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

$$\begin{cases} G_1 \parallel ... \parallel G_n \sqsubseteq_A G \\ \forall 0 \leq i \leq n. \quad A \parallel (G_1 \parallel ... \parallel G_{i-1} \parallel G_{i+1} \parallel ... \parallel G_n) \sqsubseteq_{G_i} A_i, \end{cases}$$

The above theorem specifies the proof obligations that need to be discharged in order to be able to infer dominance in Step 2 of the verification methodology described in §2.

5 A toy example

Figure 2 presents a small example of system composed of three communicating components. The notation is not fully detailed here but should be relatively straightforward. The complete version of the paper provides all the missing detail. We are interested in automata communicating by asynchronous messages. An automaton is contained within a frame, arrows between frames represent messages that are *output* by one automaton and *input* by the other. It is assumed that each automaton has an implicit variable *queue* which stores the incoming messages. The input-enabledness property of TIOA is well suited here: in any state, the automaton can *input* a message and store it in the queue; these *input* transitions are not represented in the figure.

Outputs of a message m are denoted $!m$, consumption of a message m when at the head of the queue is denoted $\downarrow m$. i, j and integer variables while x, y are clocks. We use *urgency* labels to implicitly constrains the set of trajectories starting in a state, like in TA with urgency [2] : *eager* transitions with no guard restrict the set of trajectories to point trajectories only, *eager* transitions with a clock guard restrict the set of trajectories so that they end in the point where the guard becomes true, while *lazy* transitions do not add any restrictions (time may elapse indefinitely).

The system K contains three components K_1, K_2 and K_3: K_1 sends a message a to the environment and a message p to K_2, then awaits for a q signal from K_2. If q is received before a deadline δ_1, K_1 emits a again, otherwise it goes back to the initial state when q is received. In addition, K_1 counts the number of a's emitted (in i), and can answer a message m with a message $n(i)$.

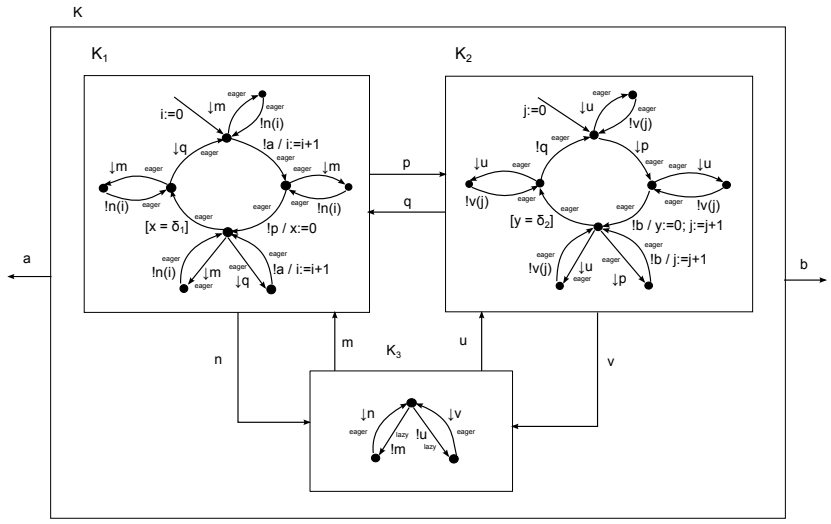

Fig. 2: A system of three communicating components.

K_2 waits for p then sends a signal b to the environment. After that, it waits for δ_2 time units and sends q to K_1; if p is received during this time, b is emitted again. K_2 counts the number of b's emitted (in j), and can answer a message u with a message $v(j)$.

K_3 sporadically sends m and u to K_1 respectively K_2.

The interesting property of this system is that, if $\delta_1 < \delta_2$, then the composition emits a sequence alternating a's and b's, represented in Figure 3. K_3 does not play any role in this property, but hinders its verification if one tries to use model-checking directly on $K_1 \parallel K_2 \parallel K_3$ as exchanges of m, n, u, v will largely contribute to the global combinatorial explosion.

Figure 4 presents a the contracts for the two components K_1 and K_2 which can be used for proving the property.In the case of K_1, the assumption over the environment sends q after at least δ_1 time units since p is received and the component guarantees that consecutive a are separated by an input of q. For K_2, the environment guarantees that it will send p only after receiving a $q$and the component guarantees a delay of δ_2 time units between an output of b and an output of q. For K_3 the contract is given by two empty automata since we don't need any assumption/guarantee on K_3 for proving φ.

The first step of the verification, as presented in Section 2, is to prove that the modelled components satisfy the given contracts. Then, in the second step we prove that the contracts $\{C_1, C_2, C_3\}$ dominate a top-level contract for the system, C, shown in Figure 5. This contract guarantees that a's and b's alternate

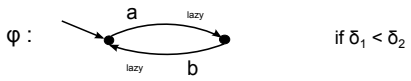

Fig. 3: The property that our example has to satisfy.

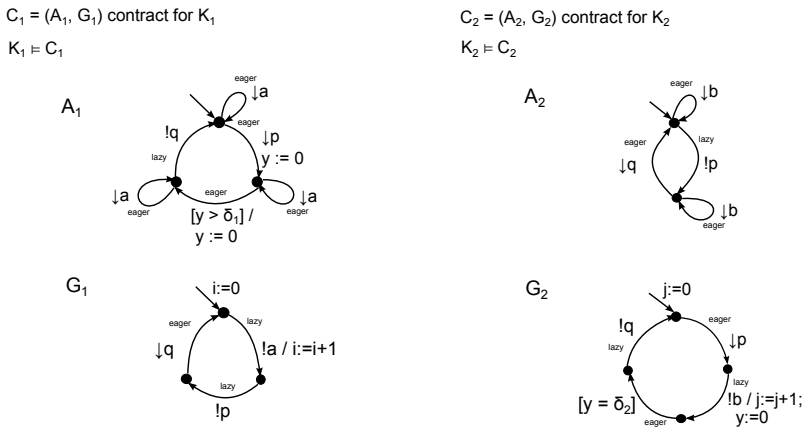

Fig. 4: Contracts for the components K_1 and K_2 of the running example.

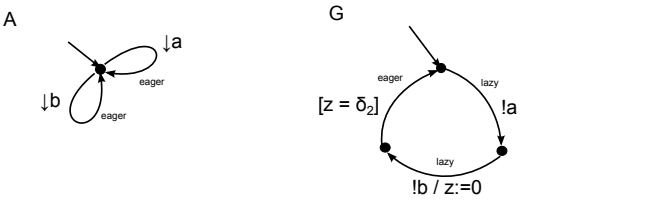

Fig. 5: Top-level contract for K.

with at least a δ_2 delay between them. No assumption is made on the environment. Verifying dominance consists in verifying several refinement under context relations: (1) $G_1 \parallel G_2 \sqsubseteq_A G$, (2) $A \parallel G_1 \sqsubseteq_{G_2} A_2$ and (3) $A \parallel G_2 \sqsubseteq_{G_1} A_1$ (note that we dropped G_3 and A_3 which are empty). The compositions involved have in principle a much smaller state space than the original system K. The last step in the verification of a system model is to prove that the top contract satisfies the global property, i.e. $A \parallel G \preceq \varphi$, which is true for the running example.

6 Conclusions and future work

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. For the moment, the method is demonstrated on a small toy example, and many steps of the method remain manual. For example, for the sake of generality, the conformance relation used in the definition of contract satisfaction and in the proof obligations for dominance is TIOA trace inclusion. However, for practical systems one can verify the existence of a simulation, which implies trace inclusion, and for which an efficient automated procedure exists [14].

In addition to other reasons for using contracts mentioned in the introduction, we believe that contract-based reasoning can potentially alleviate the problem of combinatorial explosion for the verification of large systems. Actually our

motivation for exploring contracts is driven by potential applications in system engineering using SysML [9]. In [3] we have presented a case study of an industrial-scale system and we have sketched a proof method for a core property of that system which would use contracts, but which remained to be done. The present work is a first step towards introducing contracts in SysML and providing a full solution to that problem. However, a lot of work remains to be done: define a suitable syntax for contracts in SysML, define the semantic mapping between the SysML components and contracts and their TIOA counterparts, provide the automatic verification support for contract satisfaction and dominance based on simulation checking, and finally provide quantitative evidence for the efficiency of contract-based versus direct verification .

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Information and Computation*, 163, 2000.
3. Iulia Dragomir, Iulian Ober, and David Lesens. A case study in formal system engineering with SysML. In *Engineering of Complex Computer Systems (ICECCS), 2012 17th IEEE International Conference on*, july 2012.
4. E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *ICALP*, volume 85 of *LNCS*. Springer, 1980.
5. Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata - Second Edition*. Morgan & Claypool Publishers, 2010.
6. K. Larsen, U. Nyman, and A. Wasowski. Interface input/output automata. In *FM 2006: Formal Methods*, volume 4085 of *LNCS*. Springer, 2006.
7. RTCA Inc. Software Considerations in Airborne Systems and Equipment Certification. Document RTCA/DO-178C, 2011.
8. Bertrand Meyer. Applying Design by Contract. *Computer*, 25(10):40–51, October 1992.
9. OMG. Object Management Group – Systems Modeling Language (SysML), v1.1. Available at http://www.omg.org/spec/SysML/1.1/, 2008.
10. D.L. Parnas and D.M. Weiss. Active design reviews: Principles and practices. In *Proceedings, 8th International Conference on Software Engineering (ICSE), London, UK, August 28-30, 1985*. IEEE Computer Society, 1985.
11. Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.
12. Sophie Quinton. *Design, vérification et implémentation de systèmes à composants*. PhD thesis, Université de Grenoble, 2011.
13. Sophie Quinton, Susanne Graf, and Roberto Passerone. Contract-Based Reasoning for Component Systems with Complex Interactions. Technical Report TR-2010-12, VERIMAG, 2010. http://www-verimag.imag.fr/TR/TR-2010-12.pdf.
14. Farn Wang. Symbolic simulation-checking of dense-time automata. In *FORMATS*, volume 4763 of *LNCS*, pages 352–368. Springer, 2007.