AALBORG UNIVERSITY
DENMARK

ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems

David, Alexandre; Larsen, Kim Guldstrand; Nyman, Ulrik; Legay, Axel; Wasowski, Andrzej

Published in:
Lecture Notes in Computer Science

DOI (link to publication from Publisher):
10.1007/978-3-642-15643-4_29

Publication date:
2010

Document Version
Early version, also known as pre-print

Link to publication from Aalborg University

Citation for published version (APA):
David, A., Larsen, K. G., Nyman, U., Legay, A., & Wasowski, A. (2010). ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems. *Lecture Notes in Computer Science*, *6252*, 365-370. https://doi.org/10.1007/978-3-642-15643-4_29

ECDAR: An Environment for Compositional Design and Analysis of Real Time Systems

Alexandre David[1], Kim. G. Larsen[1], Axel Legay[2], Ulrik Nyman[1], Andrzej Wąsowski[3]

[1] Computer Science, Aalborg University, Denmark
[2] INRIA/IRISA, Rennes Cedex, France
[3] IT University of Copenhagen, Denmark

Abstract. We present Ecdar a new tool for compositional design and verification of real time systems. In Ecdar, a component interface describes both the behaviour of the component and the component's assumptions about the environment. The tool supports the important operations of a good compositional reasoning theory: composition, conjunction, quotient, consistency/satisfaction checking, and refinement. The operators can be used to combine basic models into larger specifications to construct comprehensive system descriptions from basic requirements. Algorithms to perform these operations have been based on a game theoretical setting that permits, for example, to capture the real-time constraints on communication events between components. The compositional approach allows for scalability in the verification.

1 Overview

The context. Contemporary IT systems are assembled out of multiple independently developed components. Component providers operate under a contract on what the interface of each component is. Interfaces are typically described using textual documents or models in languages such as UML or WSDL. Unfortunately, such specifications are subject to interpretation. To avoid the risk of ambiguity, we recommend mathematically sound formalisms, such as interface theories, whenever possible. A good interface theory supports *refinement checking* (whether an interface can be replaced by another one), *satisfaction checking* (whether an implementation satisfies the requirements expressed with the interface), *consistency checking* (whether the interface can be implemented), a *composition operator* (structurally combining interfaces), a *conjunction operator* (computing a specification whose implementations are satisfying both operands), and a *quotient operation* that is the adjoint for composition. It should also guarantee important properties such as independent implementability [10].

It has been argued [7, 10] that *games* constitute a natural model for interface theories: each component is represented by an automaton whose transitions are typed with *input* and *output* modalities. The semantics of such an automaton is given by a two-player game: the *input* player represents the environment, and the *output* player represents the component. Contrary to the input/output model proposed by Lynch [13], this semantic offers (among many other advantages) an optimistic treatment of composition (two interfaces can be composed if there exists at least one environment in which they can interact together in a safe way) and refinement (the refined system should accepts at least the same inputs and not produce more outputs). Game-based interfaces were first developed for *untimed systems* [10, 8] and the composition and refinement operations were implemented in tools such as TICC [1] or CHIC [5].

Example. We will demonstrate our tool, ECDAR, by executing a compositional verification process. To that end we introduce a running example based on a modified real-time version of Milner's scheduler [14]. Fig. 1 (left) shows a single node, which can receive a start signal on rec_i. The node subsequently begins external work by outputting on w_i. In parallel to this it can forward the token by outputting on rec_{i+1}, but only after a delay between d and D time units. Fig. 1 (right) illustrates a ring of such nodes M_i in which some nodes have been grouped together. This grouping exemplifies a part of the specification, which we will later be able to replace with an abstraction SS_i in order to execute a compositional proof.

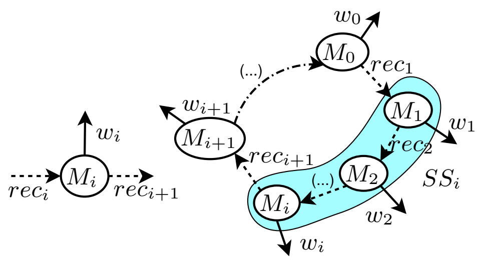

Fig. 1. Overview of Milner's scheduler example and the sub-specification SS_i.

The timed case. The above example contains timing requirements that cannot be handled with tools such as TICC or CHIC, designed for untimed systems. There exist timed formalisms but they do not provide a satisfactory notions of composition and refinement. We have recently proposed the first complete *timed* interface theory based on timed games [6]. The idea is similar to the untimed case: components are modelled using timed input/output automata (TIOAs) with a timed game semantics [4]. Our theory is rich in the sense that it captures all the good operations for a compositional design theory. In this paper we go one step further and present ECDAR, a tool that implements the theory of [6]. We thus propose the first complete game-based tool for timed interfaces in the dense time setting. ECDAR implements checkers such as satisfaction/consistency, refinement, and satisfaction of TCTL formulas. The tool also supports the classical compositional reasoning operations of conjunction and composition. To the best of our knowledge, ECDAR is the first tool to propose an implementation of quotient. In addition, it comes with a user-friendly interface, where errors are reported in an intelligible way.

2 An Integrated Environment for Design and Analysis

The user interface of ECDAR is divided into two parts: 1) the *specification interface* where automata are specified in a graphical manner, and 2) the *query interface* where one can ask verification questions.

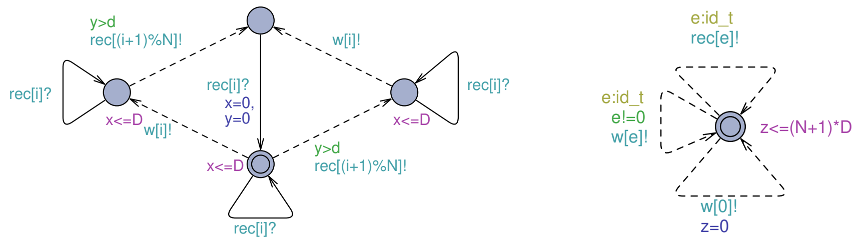

Fig. 2. Left: Template for a single node M_i. Right: Template for the overall specification.

Specification Interface. The specification interface of ECDAR uses the language of UPPAAL-TIGA to describe timed I/O automata (instead of timed game automata) with *input* and *output* modalities that are essential in this case. TIOAs communicate via broadcast channels. Global (shared) variables are not permitted. The user specifies whether the TIOA should be viewed as an implementation or as a specification. For implementations, the tool checks on-the-fly if every state respects the *independent progress* property [6], that progress must be ensured by the implementation. Details are available at ecdar.cs.aau.dk. The tool has its own engine, which reuses components of the game engine of UPPAAL-TIGA to support the new operators.

We model the scheduler using templates, in an entirely modular way. One only needs to instantiate more nodes to make a larger instance of the system. A single node of our scheduler is shown in the left side of Fig. 2. In the initial location of the specification, it is ready to receive a message on the channel `rec[i]?`. After this there are two ways to return to the initial state depending on the order in which it starts its work (`w[i]!`) and passes on the token (`rec[(i+1)%N]!`). The first node of the system M_0 is instantiated with a different initial location (the bottom-most one), reflecting the fact that it holds the token initially. The right side of Fig. 2 shows the overall specification S_0 of the system. It requires that `w[0]!` occurs at least every $(N+1)*D$ time units. Remaining actions can be executed freely.

Query Interface. The query interface provides two main checkers, the *refinement* checker and the *consistency* checker. The refinement checker is used to decide if an implementation satisfies a given specification or if a specification refines another one. As stated in [6], refinement checking reduces to solving a safety timed game between the two components. For our example one way to verify that the scheduler is correct is to verify a property of the type:

```
refinement: ( M0 || M1 || M2 || M3 || M4 ) <= S0
```

We call this type of verification monolithic, since it constructs a specification precisely representing the entire system. The tool provides a strategy to prove or disprove the property, which can be used to refine the model. The strategy can be played interactively. The consistency checker is used to check whether a specification admits at least one implementation. This question reduces to the one of deciding if there exists a strategy for the output player to avoid reaching *bad states* in the specification, i.e., states that do not satisfy the *independent progress* property. A *pruning* facility removes all the states not covered by the strategy. It can drastically reduce the state-space of the system. Following a similar principle, it is possible to constrain an interface with a TCTL* formula. For example, like in [11], one can use a Büchi objective to remove states allowing Zeno behaviours. This is the first time that a tool for compositional reasoning proposes this feature in the dense time setting.

3 Illustration and Experiment

In our example we have a ring of N nodes. It is natural to verify the monolithic property in order to show that the composed system refines the overall specification. Unfortunately, this strategy fails due to state-space explosion. As the number of components is increased, the state space grows, and more nondeterminism and interleaving is introduced in the system.

In order to combat the problem we apply compositional verification. The idea is to create N sub-specifications that are used in a series of refinement steps. First one shows that $M_1 \leq SS_1$. After this it is proved for increasing indexes, 1 to N that $SS_i || M_{i+1} \leq SS_{i+1}$. Finally the property $SS_n || M_0 \leq S_0$ is checked. Fig. 3 gives the properties for five nodes. The sub-specification aims at capturing the important aspect of the subsystem needed for the next step in the verification process of the overall property. It is very important to notice that the sub-specification is like all the other components in the system created as a template and that thus it is modelled only once and then instantiated with different indices.

```
refinement: M1 <= SS1
refinement: ( SS1 || M2 ) <= SS2
refinement: ( SS2 || M3 ) <= SS3
refinement: ( SS3 || M4 ) <= SS4
refinement: ( SS4 || M0 ) <= S0
```

Fig. 3. Incremental verification.

Here the sub-specification SS_i, as shown in Fig. 4, is a model for a sequence of nodes $M_1 || \ldots || M_i$ (see *Fig.* 1). Informally SS_i is expressed as following, noting that the relevant ports for this subsystem are `rec[1]?`, `w[e]!` `(0<e<=i)` and `rec[i+1]!`: Under the assumption that a) the time elapsing between two `rec[1]?` is more than $N * d$ time-units and b) there are no two consecutive `rec[1]?` without a `rec[i+1]!`, then it is guaranteed that `rec[i+1]!` will occur within $[i * d, i * D]$ time units from `rec[1]?`.

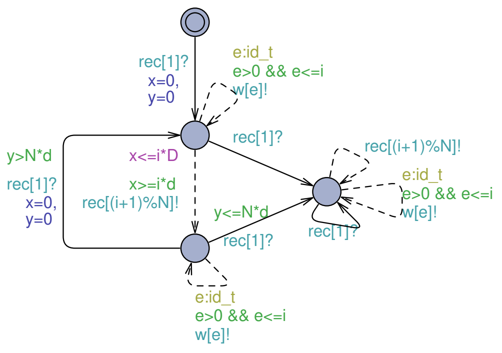

Fig. 4. The sub-specification SS_i that abstracts the the sub-system $M_1 || \ldots || M_i$.

We have performed experiments for different values of N, number of nodes in the ring, and d the minimum time delay before passing on the token. We have fixed the upper time limit for passing the token to 30. The results of the experiments are shown in Table 1. The table shows the time used to check a given property measured in seconds. For each value of N we have two rows. The top one represents the verification of all the steps in the compositional verification while the bottom row represents the verification of one monolithic property. If the verification took more than 600 seconds we stopped it. We had one instance where ECDAR ran out of memory which is indicated by `om`. The time results that are written in *italics* are the cases in which the compositional verification gave a negative result. In these cases one needs to propose more precise sub-specifications in order to make the compositional verification work. The monolithic method gives positive results in these cases.

In the case where d is close to D there is very little interleaving in the system and in this case the verification of the monolithic property is the fastest. The smaller the d value the more interleaving appears in the system and in these complex cases the compositional verification shows its strength. The cases where the compositional verification beats the monolithic are marked by **boldface**.

4 Related Work

In the untimed setting multiple contributions of Alfaro et al. focus on the operations of composition and refinement. Hence, tools such as TICC or CHIC only provide these operations. Theories exists for quotient [3] and conjunction [12], but they have not been implemented neither in TICC nor in CHIC. More recently, Bauer et al. have proposed a new extension of interface automata with

Table 1. Results of the verification experiments.

	$d = 29$	20	10	9	8	6	4
$n = 5$	0.080	0.097	0.191	*0.169*	*0.172*	*0.151*	*0.205*
monolithic	0.034	0.034	0.073	1.191	1.189	64.933	> 600
$n = 6$	0.102	0.133	0.231	*0.228*	*0.238*	*0.238*	*0.294*
monolithic	0.040	0.043	0.095	6.786	6.791	> 600	> 600
$n = 8$	0.225	0.349	0.516	**0.515**	*0.540*	*0.600*	*0.582*
monolithic	0.076	0.076	0.230	88.542	88.642	> 600	> 600
$n = 12$	0.830	1.414	1.802	**1.895**	**1.831**	*2.079*	*2.181*
monolithic	0.220	0.223	0.843	> 600	> 600	> 600	> 600
$n = 20$	4.990	9.739	12.377	**11.923**	**12.041**	**12.438**	*12.764*
monolithic	1.038	1.030	4.523	> 600	> 600	> 600	> 600
$n = 30$	22.053	45.709	55.728	**55.345**	**55.112**	**54.702**	*56.164*
monolithic	3.791	3.778	17.652	> 600	> 600	> 600	`om`

new definition for composition/compatibility and refinement; these results are implemented in the MIO Workbench [2]. This work remains at the level of untimed systems, not considering operations such as quotient or pruning.

A first dense time extension of the theory of interface automata has been developed in [11]. The theory in [11] focuses exclusively on reducing composition and consistency checking to solving timed games and does not provide any definition and algorithms for refinement, conjunction, and quotient. In [9], Alfaro and Faella proposed an efficient implementation of the algorithm used to solve the timed games introduced in [11], but for the discretized time domain only. In addition, they also proposed an extension of TICC to the timed setting. This version of TICC does not provide the same services as ECDAR does. First, timed TICC only supports consistency checking and composition; the usefulness of the tool for compositional design of real time systems is thus limited. Second, the tool does not offer a user friendly interface and the interactions with the user are extremely limited. Last, the tool works on the discretized time domain only. Hence, all the complications introduced by the dense setting are not studied.

References

1. B. T. Adler, L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, V. Raman, and P. Roy. Ticc: A tool for interface compatibility and composition. In *CAV*, volume 4144 of *LNCS*, pages 59–62. Springer, 2006.
2. S. S. Bauer, P. Mayer, A. Schroeder, and R.Hennicker. On weak modal compatibility, refinement, and the mio workbench. In *TACAS*, volume 6015 of *LNCS*, pages 175–189. Springer, 2010.
3. P. Bhaduri. Synthesis of interface automata. In *ATVA*, volume 3707 of *LNCS*, pages 338–353. Springer, 2005.
4. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, 2005.
5. Chic, 2003. http://www-cad.eecs.berkeley.edu/~tah/chic/.
6. A. David, K. Larsen, A. Legay, U. Nyman, and A. Wąsowski. Timed I/O automata: a complete specification theory for real-time systems. In *HSCC*, 2010. Accepted.
7. L. de Alfaro. Game models for open systems. In *Verif (Theory in Practice)*, volume 2772 of *LNCS*. Springer, 2003.
8. L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea. Sociable interfaces. In *FroCos*, volume 3717 of *lncs*, pages 81–105. Springer, 2005.
9. L. de Alfaro and M. Faella. An accelerated algorithm for 3-color parity games with an application to timed games. In *CAV*, volume 4590 of *LNCS*. Springer, 2007.
10. L. de Alfaro and T. A. Henzinger. Interface-based design. In *Marktoberdorf Summer School*. Kluwer Academic Publishers, 2004.
11. L. de Alfaro, T. A. Henzinger, and M. I. A. Stoelinga. Timed interfaces. In *EMSOFT*, volume 2491 of *LNCS*, pages 108–122. Springer, 2002.
12. L. Doyen, T. A. Henzinger, B. Jobstman, and T. Petrov. Interface theories with component reuse. In *EMSOFT*, pages 79–88. ACM Press, 2008.
13. N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. Technical Report MIT/LCS/TM-373, The MIT Press, Nov. 1988.
14. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.