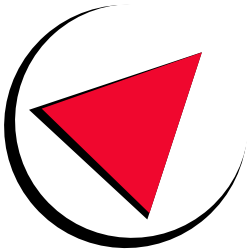

L I F C

LABORATOIRE D'INFORMATIQUE DE L'UNIVERSITE DE FRANCHE-COMTE

The tool VeSTA: Verification of Simulations for Timed Automata

Françoise Bellegarde — Jacques Julliand — Hassan Mountassir — Emilie Oudot

Rapport Technique n° RT2006–01

THÈME 2 – 13/07/06

FRE CNRS 2661

The tool VeSTA: Verification of Simulations for Timed Automata

Françoise Bellegarde , Jacques Julliand , Hassan Mountassir , Emilie Oudot

Thème 2

Techniques Formelles et à Contraintes

13/07/06

Abstract: This document presents the tool VeSTA, a push-button tool for checking the correct integration of a component into a composite timed system. Correct integration means here that already established properties of the component are preserved when it is merged into its environment. VeSTA checks this correctness by means of a so-called divergence-sensitive and stability-respecting timed τ-simulation. A successful verification of this simulation guarantees that all linear properties which can be expressed in the logical formalism MITL (in particular liveness and bounded-response) are preserved by the integration.

Moreover, the development of VeSTA was guided by the architecture of the Open-Kronos tool. This gives the possibility, as additional feature, to connect the models considered in VeSTA to the modules of the Open-Caesar verification platform.

Keywords: τ-simulation, integration of components, timed automata, preservation of linear-time properties.

Laboratoire d'Informatique de l'Université de Franche-Comté,
UFR Sciences et Techniques,
16, route de Gray, 25030 Besançon Cedex (France)
Téléphone : +33 (0)3 81 66 64 55 — Télécopie : +33 (0)3 81 66 64 50

L'outil VeSTA: Vérification de Simulation pour les Automates Temporisés

Résumé : Ce document présente l'outil VeSTA, qui permet de vérifier qu'un composant s'intègre correctement au sein d'un système temporisé à base de composants. Par intégration correcte, nous entendons que les propriétés locales d'un composant, vérifiées sur ce composant, sont préservées lorsque celui-ci est plongé dans son environnement. VeSTA vérifie cette intégration par le biais d'une τ-simulation temporisée sensible à la divergence et respectant la stabilité. En d'autres termes, VeSTA vérifie que le composant à intégrer simule le modèle obtenu après intégration. Lorsque cette simulation est établie, la préservation de toutes les propriétés linéaires (exprimables par la logique MITL) du composant est garantie.

Notons également que nous avons adopté une architecture pour VeSTA basée sur le modèle de celle de l'outil Open-Kronos. Ceci donne la possibilité de connecter les modèles considérés par VeSTA aux modules de vérification de la plateforme Open-Caesar.

Mots-clés : τ-simulation, intégration de composants, automates temporisés, préservation de propriétés linéaires.

Laboratoire d'Informatique de l'Université de Franche-Comté,
UFR Sciences et Techniques,
16, route de Gray, 25030 Besançon Cedex (France)
Téléphone : +33 (0)3 81 66 64 55 — Télécopie : +33 (0)3 81 66 64 50

Contents

1 Motivations

Component-based modeling is a more and more popular way to model complex systems. We can cite for instance timed systems which are often modeled this way. It consists in decomposing the system into a set of sub-systems, called components. Next, each component is modeled. For timed systems, a formalism often used is timed automata [AD94]. The complete composite model is obtained by putting together all these components w.r.t. some composition operator. A particularly often used operator is the parallel composition operator ||, defined as a synchronized product where synchronizations are done on actions with the same label.
From a verification point of view, two kinds of properties can be checked on these models: global properties, concerning the behaviour of the global model, and local properties only concerning the components or some assembling of components. An attractive verification method which can be used to verify these properties is model-checking. However, it is well-known that this method has the drawback to be difficult to apply on large-sized models. Nevertheless, in general, for both kind of properties, the method is performed on the global model, and thus can be difficult and even impossible to perform.
A way out for local properties would be to check them only on the components, or assembling of components, they concern. Model-checking would be here still applicable since the size of the components is generally small. Obviously, for this alternative to be valid, already established properties of a component have to be preserved on the global model, i.e., when the component is merged into its environment. With the parallel composition operator ||, the preservation of safety properties is guaranted for free. However, this is not the case for liveness or bounded-response properties.
We defined in [BJMO05] a so-called *divergence-sensitive and stability-respecting timed τ-simulation* operating between timed automata. We proved that this simulation preserves all linear properties expressed in the logical formalism MITL (Metric Interval Temporal Logic) [AFH96]. Thus, a successful checking of this simulation between two timed automata $C||Env$ and C (written $C||Env \preceq_{ds} C$) ensures that all linear properties established on C are preserved when C is integrated in its environment Env. The timed τ-simulation is defined between the traces of $C||Env$ and C, and is characterized by (1) if $C||Env$ can make an action of C after some amount of time, then C could do the same action after the same amount of time, (2) internal actions of Env, called τ, must stutter. Divergence-sensitivity ensures that internal actions τ of Env will not take the control forever and stability-respecting guarantees that the integration of C in Env will not create new deadlocks.

We developped the tool VeSTA (*Ve*rification of *S*imulations for *T*imed *A*utomata) to automate the verification of this simulation, in the framework of component-based timed models. Moreover, the architecture of the core of the tool was inspired by the one of Open-Kronos [Tri98], to make possible the connection of the models described in VeSTA to the verification platform Open-Caesar [Gar98].

This document presents how to use VeSTA and is organized as follows. Section 2 gives a global description of the tool. Section 3 presents how to create the component-based timed models considered in VeSTA. The generation of compositions between components is described in section 4. Section 5 presents an extra feature of VeSTA consisting in specifying the local properties of the components to optimize the verification of the simulations. This verification as well as its options are presented in section 6. Finally, section 7 explains how to install and run the tool.

2 Overview of VeSTA

VeSTA considers timed systems modeled in a compositional framework. It is a push-button tool for checking the correct integration of a component in its environment. A correct integration means that the local properties of the component are preserved when the component is merged into its environment. VeSTA checks it by means of the *divergence-sensitive and stability-respecting timed τ simulation* defined in [BJMO05]. Thus, VeSTA allows users to

- Create a component-based timed model,
- Give the interactions between the components,
- Optionnally, specify the local properties of the components which must be preserved,
- Merge components together by means of the classic parallel composition operator ||,
- Check simulations.

Thus, VeSTA allows to manage projects composed of all these elements: a component-based timed model, compositions, and the simulations checked.

2.1 Architecture of VeSTA

The architecture of VeSTA is shown in Fig. 1. The models considered consists of three kinds of elements: the set of components (saved in .aut files), and possibly their local properties (prop), the types of the variables used in the components and the interactions between components (sync). So-called composite components, created by parallel composition of components, can be automatically generated (.exp files). They can also have local properties. All these elements can be captured via the Graphical User Interface of VeSTA.

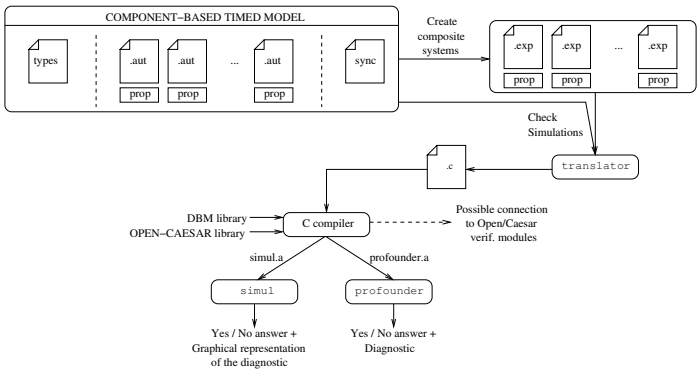

Figure 1: Architecture of VeSTA

The core of the tool consists in three modules: `translator`, `simul` and `profounder`. `translator` creates a file .c which implements data structures and functions to generate a symbolic graph (the so-called simulation graph) for each composite components involved in the simulation. Note that this file .c can be used as input to the modules of the verification platform Open/Caesar. `simul`

is the executable used to check the *stability-respecting timed τ-simulation*. The previous file .c is first compiled and linked to DMB and Open/Caesar libraries (DBM libraries permits to manipulate the timing constraints of the model). Then, `simul` is created. Finally, `profounder` is the executable used to check the *divergence-sensitive* part of the simulation. It was written by Stavros Tripakis. More informations about `profounder` and the algorithms implemented in it can be found in [TYB05] and [Tri98].

2.2 Quick overview of the Graphical User Interface

The Graphical User Interface of VeSTA is shown in Fig. 2. The tree on the left is an explorer allowing to navigate between the different elements of the model, the compositions, and the simulations checked. The bottom-right part is a log window, displaying informations such as syntax errors. The top-right part is the main element of the GUI. It consists in five tabs:

- the tab *Types* displays the different types of the variables used in the model,
- the tab *Interactions* shows the interactions between the components,
- the tab *Basic Components* contains all the components of the model,
- the tab *Composite Components* contains all the compositions already created between the components of the model,
- the tab *Simulations* contains the results for each already checked simulation.

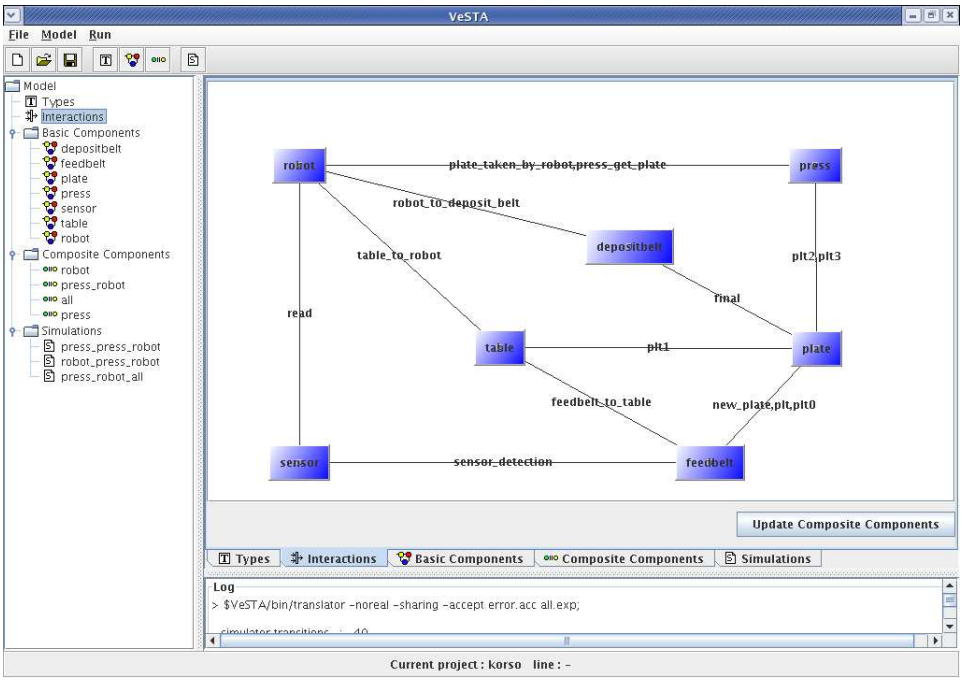

Figure 2: Overview of VeSTA Graphical User Interface

Menus and toolbar. The `File` menu presents the usual possibilities for creating a new project (`New`), opening an existing project (`Open...`), saving the current project (`Save`) and exiting the application (`Exit`). The three first items are also represented as buttons in the toolbar (the three first buttons correspond respectively to the three first menu items). Note that, for the moment, a VeSTA project is saved into several files (the main project file has extension .ves), and that a missing file can prevent a further opening of the project. In the future, we intend to use a single xml file to save a project in order to avoid such situations.

The `Model` menu concerns the model. It offers the possibility to import types from an existing file (`Import types from file...`), add components to the model (`Add basic component...`), and create compositions between components, called composite components (`Create composite components...`). As for the `File` menu items, these three `Model` menu items can be accessed by the tool bar, with the fourth, fifth and sixth buttons of the toolbar.

Finally, the `Run` menu only contains one item, `Run simulation check`, giving the possibility to check simulations. The last button in the toolbar corresponds to this menu item.

3 Creating the model

This section presents how to create the models considered in VeSTA. The first step consists in creating each component of the model and giving the types of the variables used in the components. Then, one has to specify the interactions between these components.

3.1 Creating the components

Adding a basic component. Adding a new component to the model can be done via the menu item `Add Basic Component...` or by clicking on the corresponding button in the toolbar. As shown in Fig. 3, a dialog window appears giving two possibilities: either creating a new component or importing an existing component. An existing component must be contained in a file with extension .aut.

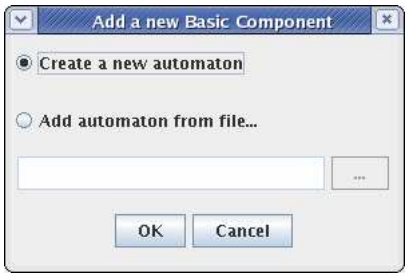

Figure 3: Adding a new Basic Component

Displaying the basic components. Each component is added in the *Basic Components* tab. The components considered in VeSTA are extended timed automata, written using the SMI library syntax. Fig. 4 shows an example of timed automaton written with this syntax.

```
xd :  clock
db_sta :  STATE

des(0,3,3)

(* Timed automaton *)

(* 0 *)
(0, robot_to_deposit_belt xd:=0 db_sta:=move, 1)
(* 1 *)
(1, [xd<=8] move db_sta:=to_take, 2)
(* 2 *)
(2, final db_sta:=wait, 0)
(* Time-progress conditions *)
[0, db_sta=wait]
[1, db_sta=move / xd<=8]
[2, db_sta=to_take]
```

Figure 4: An example of Timed Automaton, written in SMI syntax

Briefly, the first step consists in declaring each variable used in the automaton. The instruction `des(0,3,3)` means that the initial state is the state 0, and that the automaton contains 3 transitions and 3 states. Then, each transition is specified by giving the source state, the guard (facultative) and the label of the transition, updating variables (facultative), and finally giving the target state. For each state, time-progress conditions can also be specified, as well as the expected value for the variables. To get more detailed informations about this syntax, please consult the SMI web site:

`http://www-verimag.imag.fr/~async/SMI`

All the components of the model are accessible in the *Basic Components* tab, as shown is Fig. 5. Each component is displayed in a corresponding tab, where the top part of the tab contains the specification of the component and the bottom part allows to give local properties of the component. This feature is explained in details in section 5.

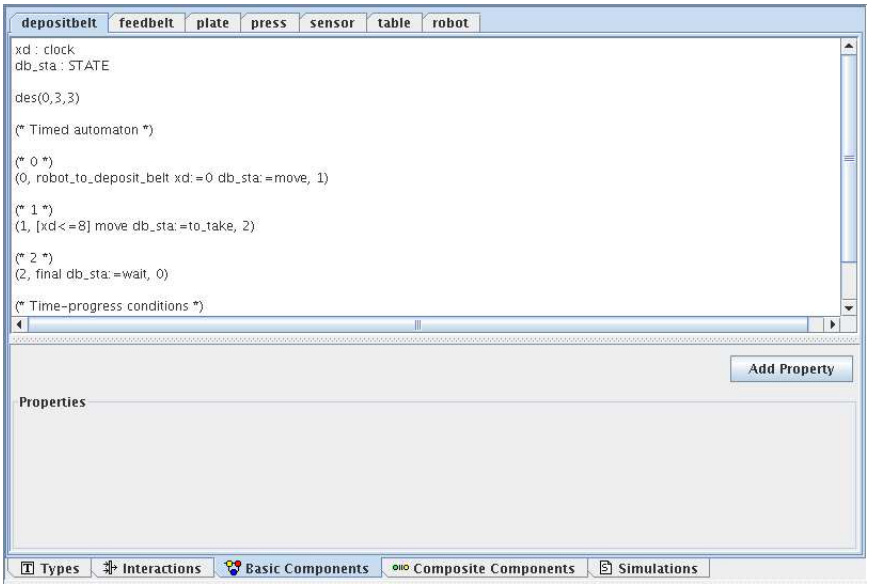

Figure 5: The *Basic Components* Tab

Renaming and/or deleting a component. When a new component is created, a name is given to it by default. To rename a component, right-click on the component in the tree on the left part of VeSTA. A popup menu appears allowing to rename the node, but also to delete it, as shown in Fig. 6.

Figure 6: Renaming and/or deleting a component

3.2 Giving the types of the variables used in the components

As for components, the syntax used for the types is the SMI syntax. There already exists some predefined types in SMI:

- `bool` for boolean variables,
- `nat` for integer variables,
- `clock` for clock variables.

User can also defines enumerate types, which must be defined in the *Types* editor of VeSTA, as shown in Fig. 7. Note that, when `nat` or `clock` variables are used, it is necessary to define an upper bound for these variables in the *Types* editor. In the example above, this bound is set to 100 thanks to the declaration `nat`{100}.

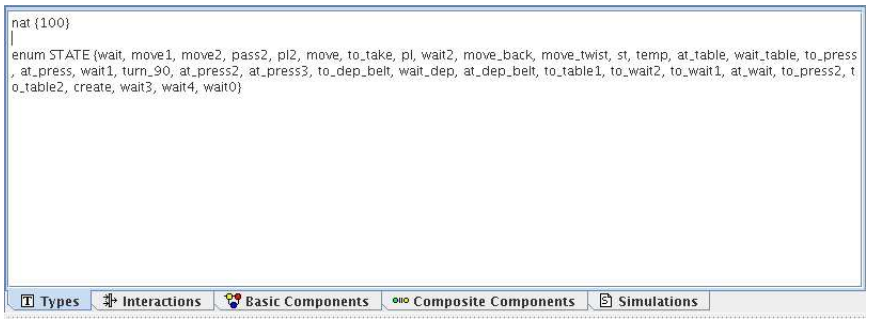

Figure 7: The *Types* Tab

3.3 Specifying the interactions between components

The last step in the definition of the model is the definition of the interactions between the components. The components interact by synchronizing on common actions. The interactions can be given by the user via the *Interactions* tab of VeSTA. As shown in Fig. 8, VeSTA uses a *graph* to draw these interactions. Each node of the graph represents a component, and a line between two components means that these components interact by synchronizing on the actions which label the line.

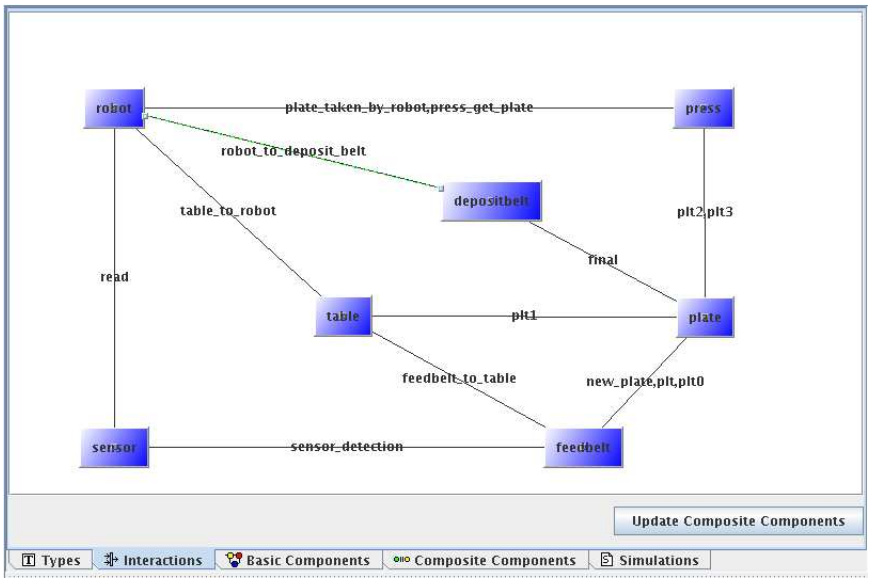

Figure 8: The *Interactions* Tab

The components are automatically added in the graph when a new component is created in VeSTA. Creating an interaction between two components is done by clicking in the center of one component, and next in the center of the second component. The line is then drawn. Its label can be given by double-clicking on it. If this label consists in more than one action, the actions must be separated using commas. A line can be suppressed by selecting it and typing the *delete* key.

The interactions given in this graph will be used to create compositions between the components. The button `Update Composite Components` in the bottom of the tab allows to update the compositions which are already created if the interactions are modified.

4 Generate composite components

Once the model is created, VeSTA offers the possibility to automatically generate compositions between chosen components, w.r.t. the given interactions. The operator considered for composition is the parallel composition operator, written ||. Informally, it is defined as a synchronized product, where the synchronizations are done on actions with the same label (the ones which are specified in the *Interactions* tab), while other actions interleave, and time elapses synchronously for each component involved in the composition.

Choosing the components. The compositions can be created from the `Create Composite Components` menu, or via the corresponding button in the toolbar. As shown in Fig. 9, a dialog window appears allowing to select components from which the composition will be created.

Figure 9: Choosing components to create a composite component

Displaying the composite components created. The composite components created this way appears in the *Composite Components* tab. As for the basic components, each composite component is displayed in a corresponding tab. The top part of the tab gives a textual representation of the composition (still in SMI syntax), while the bottom part allows to specify its local properties (see section 5). Fig. 10 shows the *Composite Components* tab.

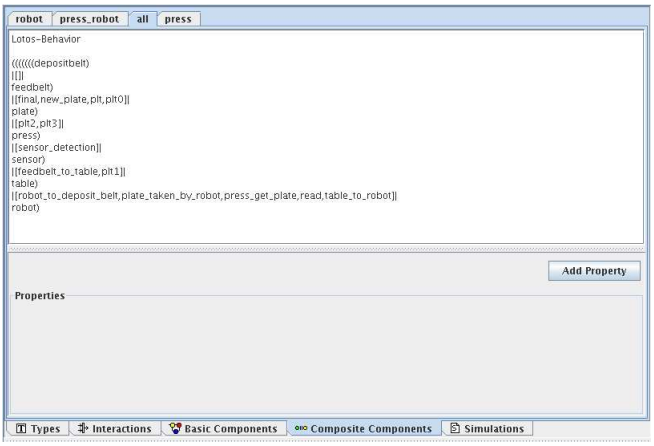

Figure 10: Choosing components to create a composite component

Renaming / deleting composite components. As basic components, composite components can also be renamed or deleted, by right-clicking on them in the tree on the left of the GUI.

5 Specifying local properties for basic / composite components

VeSTA offers the possibility to specify the local properties which has to be preserved after integration of a basic/composite component in an environment. For the moment, only response properties of the form $\Box(p \Rightarrow \Diamond q)$ can be specified. They are called *Response 1* in VeSTA.
Local properties are specified in the bottom part of the tab corresponding to a basic or composite component. To add a property, click on the button `Add property`. A new line appears in the bottom part. The list on the left allows to choose the kind of property to add. Then, a formula appears in the text field on the right. For instance, for *response 1* properties, the formula is

```
[](p => <>q)
```

User has next to replace the variables appearing in the formula (p and q in the above example) by expressions written using SMI syntax. The property can also be removed by clicking on the `Remove` button. Fig. 11 shows examples of local properties specified for a basic component.

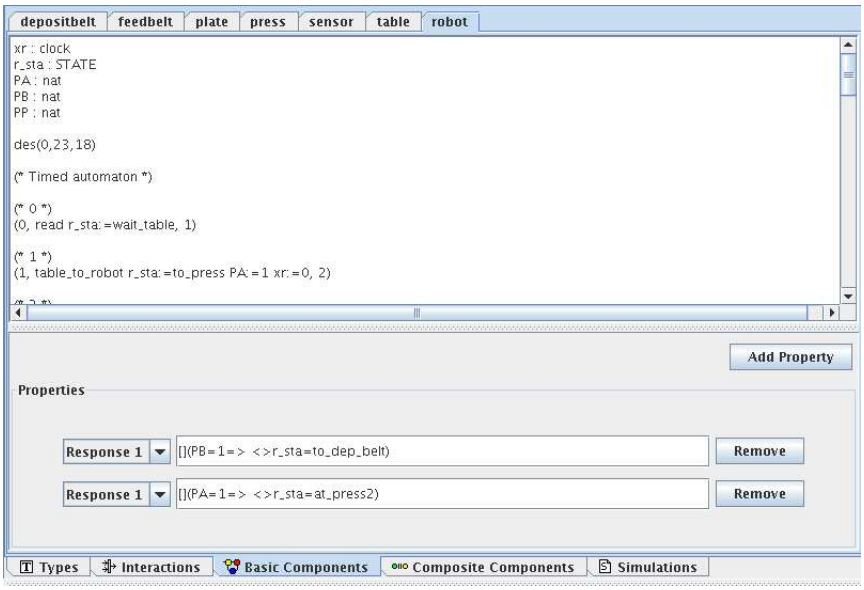

Figure 11: Specifying local properties of a basic/composite component

6 Checking simulations

The main feature of VeSTa is to check the correct integration of a (basic/composite) component C in its environment Env. This integration is checked by means of a *divergence-sensitive and stability-respecting timed τ-simulation*. To check the simulation, click on the menu item `Run simulation check...` or on the corresponding button in the toolbar. A dialog window appears to set the options of the simulation checking, as shown in Fig. 12.

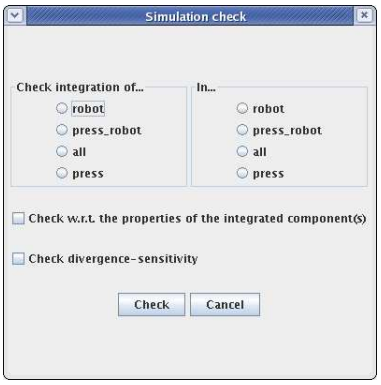

Figure 12: Setting the options of the simulation check

6.1 Simulation checking options

The first step is to choose the components for which simulation will be checked. For this, choose a component C on the left of the dialog window, and the composite component $C||Env$ in which it is integrated. Note that only composite components appear in this window. Thus, basic components must be *transformed* into composite components to be available for simulation checking. This can be done by creating a composite component only consisting in one basic component. Once the two composite components are chosen, it is possible to check the simulation, by clicking on the button `Check`. At this moment, the executable `simul` is created, and the *stability-respecting timed τ-simulation* will be checked.

The option `Chech divergence-sensitivity` allows to create the executable `profounder` to check the *divergence-sensitive* part of the simulation. To launch this verification, internal actions of the environment Env must be clearly specified in all the basic components involved in $C||Env$. A boolean variable `tau` must be declared (in the first component involved in C), and set to `true` on the transitions representing internal actions of the environment Env or to `false` on the transitions corresponding to actions of C (i.e., on all transitions of C). The divergence-sensitivity verification will then search for cycles only containing the internal actions.

The option `Check w.r.t. the properties of the integrated component(s)` allows to verify the simulation only for the local properties given for the composite component C (this also takes into account the local properties given for the basic components involved in this composition). This option allows to optimize the simulation checking, by performing the verification only on the

paths of $C||Env$ concerning the specified local properties. In return, only the preservation of these properties can be ensured.

6.2 Results of the simulation checking

For each simulation checked, a new tab is created in the *Simulations* tab of VeSTA. Two kinds of results can appear after a simulation checking: either the verification is successful, or it failed. In the first case, a message is displayed in the tab corresponding to the simulation, giving informations about the verification time. Fig. 13 shows this case.

Figure 13: Successful simulation checking

Figure 14 shows what is displayed when the verification failed. In this case, a message explaining the error is displayed in the simulation tab, as well as a graphical diagnostic.

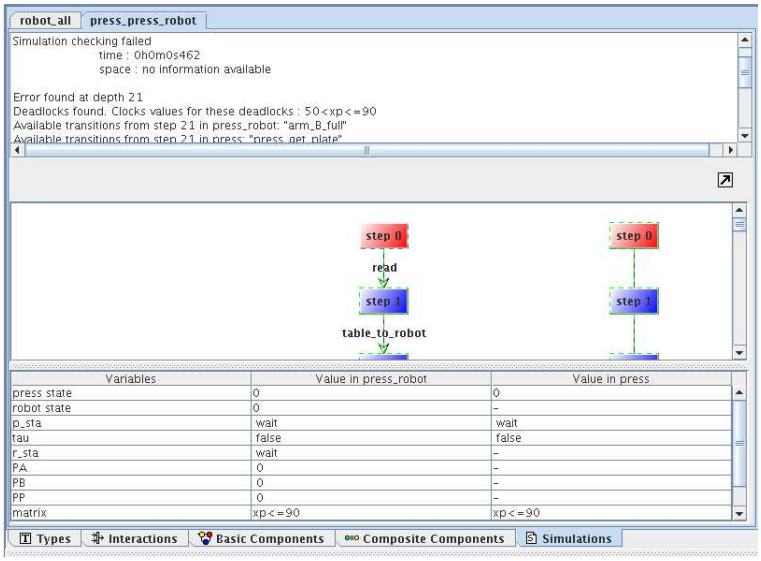

Figure 14: Diagnostic for a failed simulation checking

This graphical diagnostic consists in a trace of $C||Env$ which is not simulated by any traces of C, and the trace of C to which it had to correspond. The two traces are displayed as sequences of

states and transitions. The trace of $C||Env$ is on the left, while the trace of C is on the right. Note that some transitions do not have any label in the trace of C (these transitions are simple lines, instead of arrows). This means that the source and target states of these transitions are the same.

The table at the bottom of the diagnostic indicates the values of the variables for each state of the traces. When a state is selected by clicking on it, the values of the variables in this state are displayed in the table, as well as the timing constraints (in the `matrix` variable).
Note that, for convenience, it is possible to open the diagnostic in a new window, by clicking on the button containing a diagonal arrow.

6.3 Tricks

Running a simulation checking creates the files presented in section 2.1 which compose the core of the tool: the file .c containing a symbolic representation of the two composite components for which simulation is verified, and the two executables `simul` and `profounder`. These elements are only deleted when the project is saved or when a new simulation is checked. Thus, to gain memory for the verification, it is possible to close the GUI of VeSTA, and manually launch the executables. They can be found in the directory where the project has been saved.

7 Installation notes

VeSTA is available for Linux systems at the following URL :

`http://lifc.univ-fcomte.fr/~oudot/VeSTA`

It was developped using Java and C languages. The GUI of VeSTA is written for Java VM 1.5. The core of the tool (`translator`, `simul` and `profounder`) was tested with the compiler `gcc 3.3.2`. A good functioning of the tool is for the moment only ensured with these versions.

To install VeSTA, uncompress the file VeSTA.tar.gz. For this, open a terminal window and type the following command:

```
tar -xzvf VeSTA.tar.gz
```

A directory VeSTA is created, with the following elements:

- VeSTA.jar: the file to be run to launch the Graphical User Interface,
- Images: directory containing the images used by the GUI,
- org: directory of the package jgraph.jar[1], used to draw graphs in the GUI,
- bin: directory containing the executable `translator`,
- include: directory containing SMI libraries,
- lib: directory containing VeSTA libraries,
- Matrix: directory containing files .h of the Matrix library,
- Examples: two examples provided to test VeSTA. The first one is the well-known example of the railroad crossing, and the second one is a case study concerning the modeling of a production cell [Bur03]. This second example shows a case when the verification of the simulation fails: when checking the integration of *press* in *all*.
- documentation.pdf: this user manual.

You have to create a variable `VeSTA` in your `bashrc`, containing the path to the directory VeSTA which has been created. CADP is also needed to run VeSTA, thus make sure that CADP is installed, and that a variable `CADP`, containing the installation path of CADP, exists in your `bashrc`.

Then, run VeSTA by typing the following command:

```
java -jar VeSTA.jar
```

[1] `http://www.jgraph.com`

Credits

This work was realized at the
Laboratoire d'Informatique de l'Université de Franche-Comté (LIFC)
16 route de Gray
25030 Besancon Cedex, France
Tel: +33 (0)3 81 66 64 55
Fax: +33 (0)3 81 66 64 50
Web: `http://lifc.univ-fcomte.fr`

By Emilie Oudot, directed by Francoise Bellegarde, Jacques Julliand and Hassan Mountassir. We would like to thank Stavros Tripakis for having sent to us the distribution of Open-Kronos, and particularly the executable `profounder` and some (useful!) source files. Thanks also for the time spent answering questions.

Contact: {oudot,bellegar,julliand,mountass}@lifc.univ-fcomte.fr
Web: `http://lifc.univ-fcomte.fr/~oudot/VeSTA`

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[BJMO05] F. Bellegarde, J. Julliand, H. Mountassir, and E. Oudot. On the contribution of a τ-simulation in the incremental modeling of timed systems. In *Proceedings of the 2nd International Workshop on Formal Aspects of Component Software (FACS'05)*, pages 117–132, Macao, Macao, October 2005. Accepted for publication in ENTCS, Elsevier. To appear.

[Bur03] A. Burns. How to verify a safe real-time system: The application of model-checking and timed automata to the production cell case study. *Real-Time Systems Journal*, 24(2):135–152, 2003.

[Gar98] H. Garavel. OPEN/CAESAR: An Open Software Architecture for Verification, Simulation and Testing. In Bernhard Steffen, editor, *Proceedings of 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, Lisboa, Portugal, March 1998.

[Tri98] S. Tripakis. *The analysis of timed systems in practice*. PhD thesis, Universite Joseph Fourier, Grenoble, France, December 1998.

[TYB05] S. Tripakis, S. Yovine, and A. Bouajjani. Checking Timed Büchi Automata Emptiness Efficiently. *Formal Methods in System Design*, 26(3):267–292, May 2005.

Laboratoire d'Informatique de l'université de Franche-Comté
UFR Sciences et Techniques, 16, route de Gray - 25030 Besançon Cedex (France)

LIFC - Antenne de Belfort : IUT Belfort-Montbéliard, rue Engel Gros, BP 527 - 90016 Belfort Cedex (France)
LIFC - Antenne de Montbéliard : Centre de développement du multimédia, cours Leprince-Ringuet - 25201 Montbéliard Cedex (France)

http://lifc.univ-fcomte.fr