Verification results feedback for FIACRE intermediate language

Faiez Zalila
Toulouse University
Toulouse, France
faiez.zalila@enseeiht.fr

Xavier Crégut
Toulouse University
Toulouse, France
xavier.cregut@enseeiht.fr

Marc Pantel
Toulouse University
Toulouse, France
marc.pantel@enseeiht.fr

Abstract

Model verification through a translational semantics is experienced to check safety and liveness properties of software architectures with some Domain-Specific Modelling Languages (DSML) like for example UML, AADL, SysML or SDL in order to reuse model-checking tools. This approach has two barriers: (1) the semantic gap between the two areas, and (2) feedback and interpretation of verification results at the DSML level. Some intermediate languages, like for example FIACRE, are proposed in order to reduce the important semantic gap and to facilitate the definition of translational semantics. Nevertheless, feedbacks are still to be done. In this paper, we address the interpretation of results generated by the model checking tools and their feedback at the intermediate language level (FIACRE in our case) as scenarios that will be the basis to leverage the feedback at the DSML level.

keywords: Domain specific modeling languages, FIACRE intermediate language, feedback verification results

1 Introduction and related works

Design of critical embedded software, using high level domain-specific modelling languages (DSML), is becoming increasingly complex. The need for safety development becomes a stringent requirement. Model checking is recommended as one of the solutions to formally verify finite-state real-time embedded systems. To support it, a specific verification toolchain is often developed for each couple of modelling language / formal tool. In DSML area, [8] focuses on how to map DSML definitions (abstract syntax, concrete syntax, static semantics and behavioral semantics of a language) into Alloy, a structural language based on first-order logic, which provides effective techniques for model checking. In [12], Pelliccione *et al.* present a software tool platform for the model-based design and validation of software architectures, named CHARMY, that offers an extension called SASIM deriving from Theseus approach in [9]. Both translate the violation trace from SPIN model checker to a generated sequence diagram and an animated UML state diagrams. vUML [11] also use the same approach. All these works are based on a very ad hoc approach.

The important semantic gap between DSML and formal levels is the first barrier to designers. The FIACRE language [3] – designed as part of the TOPCASED project [4] and developed in the Quarteft project – offers a solution to simplify the development of these verification toolchains. Indeed, FIACRE provides high level constructs to make easier the translation from DSML model. Furthermore, it is an intermediate language that allows to target several verification tools (TINA[1] or CADP[2]). The toolchain can then be split in a specific part from the DSML to FIACRE and a generic and reusable one from FIACRE to one of the supported verification toolbox. The second barrier concerns the feedback of verification results. They are obtained on the formal level and have to be leverage to the domain level. The FIACRE intermediate language is again a way to split feedbacks from FIACRE intermediate model to the DSML model preceded by a low-level one from

[1] http://homepages.laas.fr/bernard/tina
[2] http://www.inrialpes.fr/vasy/cadp/

verification tools to FIACRE level. In this paper, we concentrate on the latter one and more precisely on feedbacking from TINA toolbox to FIACRE language.

The paper is organized as follows. In Section 2, we present the FIACRE language using an illustrative program and show formal verification results obtained using TINA toolbox. Section 3 introduces various steps to generate FIACRE scenario from verification results. Last section contains concluding remarks and perspectives.

2 FIACRE and behavioral extensions

FIACRE is a (French[3]) acronym for an Intermediate Format for Embedded Distributed Components Architectures. It is a formal specification language to represent both the behavioral and timing aspects of real-time systems [4]. The FIACRE language, formally described in [3], is composed of two syntactical constructs, processes and components. A process describes the behavior of sequential components. It is defined by a set of control states, each associated with an expression that specifies state transitions. Expressions are built from deterministic constructs available in classical programming languages (assignments, conditionals, sequential composition, . . .), non-deterministic constructs (choice and non-deterministic assignments), communication events on ports, and transition to next state.

A component describes the composition of processes, possibly in a hierarchical manner. It is defined as a parallel composition of components and processes communicating through ports and shared variables. Components allow to restrict the access mode and visibility of shared variables and ports.

FIACRE supports two of the most common communication paradigms: communication through shared variables and synchronization through (synchronous) communication ports. In the latter case, it is possible to associate time and priority constraints to communication over ports. The use of timing constraints is illustrated in the example of Listing 1 which models the operation of a simple manufacturing plant. A factory builds products from a command line. There are two kinds of machines available *M1* and *M2*. A line (an instance of *Linebehavior* process in the *Main* component, line 39 in Listing 1) uses just one machine (an instance of *Machine* process, line 42). A worker (an instance of *Worker* process, line 37) operates the line. The factory is subject to operational and legal requirements: (1) during their work, workers should have 5 minutes pauses (line 21 in Listing 1); (2) the duration of a machine task does not exceed 5 minutes (time interval on line 34). A Real-time extensions for the FIACRE language is proposed in [7]. It allows the specification of real-time properties. Line 4 of Listing 1 is a property based on the **leadsto** pattern for the following requirement: "the first instance of the *Worker* process must *pause* before *5* minutes of *work*".

FRAC is a compiler which generates a Time Transition System (TTS) – a generalization of Time Petri Nets (TPN) with data variables and priorities – from the states of the processes and timed transitions of the FIACRE program. pFrac[4] (Frac with real-time specification patterns) is a prototype of FRAC compiler, which takes in account real-time properties. It is based on the use of observers in order to transform the verification of timed patterns [2]. The TINA verification toolbox [5] offers several tools to work with TTS descriptions. For instance, for verification purposes, TTS specifications can be used by SELT – a model-checker for a State-Event version of LTL. Applied on the factory program, the property is found to fail and a scenario is generated as a sequence of fired transitions:

```
Linebehavior_1_t0_Worker_1_t0$0
Machine_1_t0_Linebehavior_1_t1$0
_ERROR_1$5
```

[3]FIACRE stands for Format Intermédiaire pour les Architectures de Composants Répartis Embarqués.
[4]http://homepages.laas.fr/~nabid/pfrac.html

```
type machinename is union M1 | M2 end
type linename is union line1 | line2 end

property Worker_1_sWork leadsto Worker_1_sPause inlessthan 5

process Linebehavior[LStartline: linename, LStartMachine: machinename,
                     LEndMachine: machinename, LEndline: linename]
                    (ln: linename, mn: machinename) is
  states Idle, Startmachine, WaitEndmachine, NextMachine, Restart
  from Idle LStartline!ln; to Startmachine
  from Startmachine LStartMachine!mn; to WaitEndmachine
  from WaitEndmachine LEndMachine?mn; to NextMachine
  from NextMachine wait [0,0]; to Restart
  from Restart LEndline!ln; to Idle

process Worker[WStartline: linename, WEndline: linename]
              (linepermis: linename) is
  states Idle, Work, Pause
  from Idle WStartline?linepermis; to Work
  from Work WEndline?linepermis; to Pause
  from Pause wait [5,5]; to Idle

process Machine[MStartMachine: machinename, MEndMachine: machinename]
               (nm: machinename, linepermis: linename) is
  states Idle, Work
  from Idle MStartMachine?nm; to Work
  from Work MEndMachine!nm; to Idle

component Main is
  port
    Startline: linename in [0,0],
    Endline: linename in [0,0],
    StartMachine: machinename in [0,0],
    EndMachine: machinename in [0,5]
  par * in
    par * in
      Worker[Startline, Endline] (line1)
      ||
      Linebehavior[Startline, StartMachine, EndMachine, Endline] (line1, M1)
    end
    ||
    Machine [StartMachine, EndMachine](M1, line1)
  end Main
```

Listing 1: A factory example in FIACRE

The scenario shows two transitions fired at time 0 and an *error* transition fired at time 5. The first transition *Linebehavior_1_t0_Worker_1_t0* explains the start-up of production line (*line1*) by the worker who enters into *Work* state. The second transition *Machine_1_t0_Linebehavior_1_t1* shows the use of the machine (*M1*) by the previously started line. The *error* transition is not part of the FIACRE program. It has been added by Pfrac tool as an observer. It can only be fired after 5 time units and has a higher priority than the other competing transitions. The firing of the *error* transition means that the first instance of *Worker* process has spend 5 time units in the *Work* state and thus the property failed. Obviously, this scenario is not easy to understand for the designer. So we have to feedback verification results at the FIACRE level and then at the DSML level. Our approach consists in generating a FIACRE scenario from the previous TPN scenario.

3 Feedback verification results

One purpose is to provide a scenario at the FIACRE level which could be then leveraged at the DSML level [6]. Unfortunately, FRAC does not provide traceability information required to feedback verification results. However, we can retrieve them by consulting intermediate steps during FIACRE program compilation.

Indeed, frac can generate an intermediate textual format that presents an hybrid TTS, named *"instantiated* Fiacre *"*. It contains TPN specifications (transitions, states, priorities, ...) and data processing (guards, assignments, ...) and can thus be used to generate traceability links between TTS and the original Fiacre program. Listing 2 shows a part of the instantiated Fiacre for factory program. We choose to show the corresponding transitions in the generated TPN scenario: The first one explains the *Linebehavior_ 1_ t0_ Worker_ 1_ t0* transition which has a label named *Main_ 1_ pStartline* that corresponds to a synchronisation on *Starline* port in the *Main* component. It has also a "`from` statement" that contains the changed places *Linebehavior_ 1_ sIdle* and *Worker_ 1_ sIdle* which matches respectively changed states, *Idle* state in the first instance of *Linebehavior* process and *idle* state in the first instance of *Worker* process in the Fiacre program. Next, it contains an "`assignment` statement" which describes the data processing in the TTS description. After, it explains entered places which correspond to entered states in the Fiacre program. Finally, a time interval is associated with each instantiated Fiacre transition. The instantiated Fiacre represents an information that can be used as a source to generate traceability links. It shows an intermediate artifact, generated during compiling the Fiacre program, which represents a required information to find appropriate elements (component, process, instance, state, variable, etc.) in the Fiacre level. It contains also all TPN elements (transitions, places, labels, etc.). These features allow to create links between both levels and combine TPN elements with Fiacre ones. Instantiated Fiacre artifact involves a set of Instantiated Fiacre transitions which have the same syntax than the Fiacre ones: a change state statement (`from` statement), followed by data processing statements (assignments, conditional statements, etc.), ended up with enter state statement (`to` or `loop` statement).

```
 1    Trans::Linebehavior_1_t0_Worker_1_t0: Main_1_pStartline & Main
 2      from Linebehavior_1_sIdle Worker_1_sIdle
 3       Worker_1_vlinepermis := Linebehavior_1_vln;
 4       to Linebehavior_1_sStartmachine Worker_1_sWork
 5      in [0,0]
 6
 7    Trans::Machine_1_t0_Linebehavior_1_t1: Main_1_pStartMachine & Main
 8      from Linebehavior_1_sStartmachine Machine_1_sIdle
 9       Machine_1_vnm := Linebehavior_1_vmn;
10       to Linebehavior_1_sWaitEndmachine Machine_1_sWork
11      in [0,0]
```

Listing 2: Part of the Instantiated Fiacre generated from factory program

This particularity reduces efforts to manipulate the instantiated Fiacre in order to use it on the establishment of traceability links. Manipulating an intermediate step generated during compiling can be an appropriate way because it allows to capture the same progress report for each Fiacre program. The corresponding tooling has been implemented with xText [1]; a textual grammar is defined to parse the instantiated Fiacre. It is inspired by the Fiacre one. Thanks to naming conventions, traceability to Fiacre elements is made possible. So, through an ATL transformation [10], links are added between the Fiacre and TPN levels to generate a traceability model named "linked Fiacre ". Its metamodel is similar to the instantiated one (generated by xText). It is enriched with references towards TPN and Fiacre appropriate elements. The linked Fiacre is seen as supporting traceability model in order to feedback verification results.

Our Goal is to use this traceability information with the generated scenario from selt model checker in order to generate a Fiacre one that is a succession of events. A part of the Fiacre scenario is shown in the Figure 1. To generate the Fiacre scenario, we have to find the corresponding transition in the instantiated Fiacre model from the transition in the TPN scenario. For example, the first event in the TPN scenario (*Linebehavior_ 1_ t0_ Worker_ 1_ t0*) corresponds to the first instantiated Fiacre transition in the instantiated Fiacre model (line 1 in Listing 2). Next, if this transition has a label

(*Main_ 1_ pStartline*), it is a synchronisation on the corresponding port (*Startline*) in the component (*Main*). This synchronisation event produces other port events on process instances (SynchronisationEvent, ReceiveEvent or SendEvent). `From` and `To` statements generate state events (ChangingEvent or EnteringEvent) on process instances. Finally, between them, assignment statements produce changing value of variables (VariableEvent). The time shown in the TPN scenario is the date of firing transition. A second ATL transformation generates the Fiacre scenario from the traceability model (linked Fiacre) and the TPN scenario. Figure 1 shows Fiacre events generated from the second transition in TPN scenario. for each event, we associate the time generated in the TPN scenario on the corresponding transition, one or many concerned instances and the node which defines these instances (component or process declarations).

time	EventType		Instance	Node
0	SynchronisationEvent	port: StartMachine	Linebehavior, Machine	Main
0	ChangingEvent	state: Startmachine	Linebehavior	Linebehavior
0	SendEvent	port: LStartmachine, exp: mn	Linebehavior	Linebehavior
0	EnteringEvent	state: WaitEndmachine	Linebehavior	Linebehavior
0	ChangingEvent	state: Idle	Machine	Machine
0	ReceiveEvent	port: MStartMachine, pattern: nm	Machine	Machine
0	EnteringEvent	state: Work	Machine	Machine

Figure 1: Part of a generated Fiacre scenario

We rely on the architecture defined in TopCased for executable DSML– and more precisely on the runtime events that trigger evolution on one model – to record traceability links between the domain level and the formal level. Figure 2 shows the mapping between Fiacre and TPN levels. On the TPN side, there is just one event that is *FireTransition-Event* that corresponds to several Fiacre events on the Fiacre side. These extensions allow to capture Fiacre events. Generating the Fiacre scenario is a step to make the Fiacre-TPN chain transparent. So one has only to concentrate on the DSML and Fiacre back and forth mappings without having to deal with model-checker specific formal languages. It only relies on Fiacre.

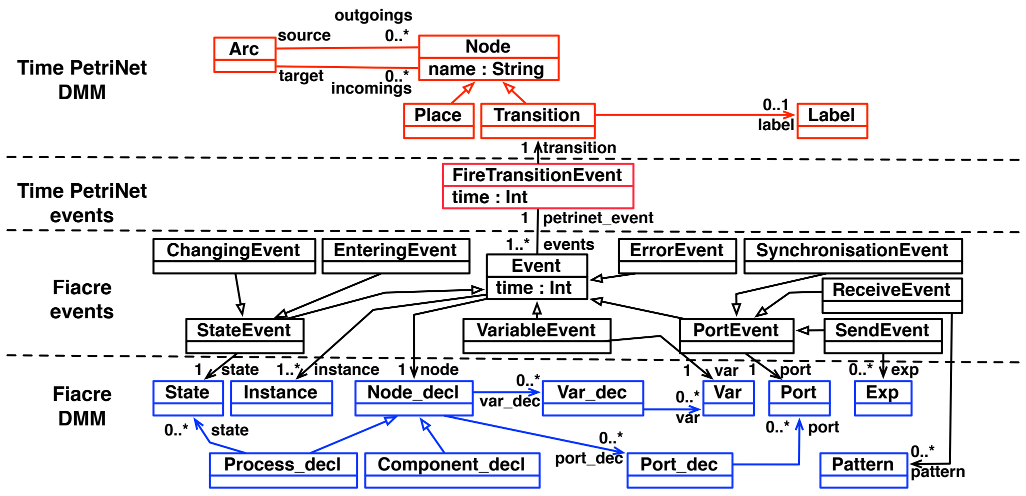

Figure 2: Event-based mapping between Fiacre and TPN metamodels

4 Conclusion and perspectives

In this paper, we have presented the current state of the feedback of verification results: we are able to construct a Fiacre scenario from the one generated by formal tools (Tina toolbox) thanks to the building of a traceability model. The proposed mapping of Fiacre domain and its events with the formal domain and its events (TPN in our case) can be adapted if we want to change formal techniques and to use others tools like for example CADP toolbox. This change has been preceded by an extension which captures possible events in the formal side. We are now working at leveraging it at the DSML level, so that it can be used by model animators for example and thus made understandable to the DSML users. This approach allows DSML experts to verify their DSML behaviors using existing formal techniques and tools without the need to manipulate them; They should just interact with the intermediate level which is more easier than the formal one.

References

[1] Tutorials and documentation for xtext 2.0. http://www.eclipse.org/Xtext/documentation/.

[2] N. Abid, S. Dal Zilio, and D. Le Botlan. Verification of Real-Time Specification Patterns on Time Transition Systems. Technical Report LAAS n.t 11365, 2011.

[3] B.Berthomieu, J.P.Bodeveix, M.Filali, H.Garavel, F.Lang, F.Peres, R.Saad, J.Stoecker, and F.Vernadat. The syntax and semantics of fiacre - draft version 3.0. LAAS Reports 07264, LAAS, 36p., 2011-01-14.

[4] B. Berthomieu, J-P. Bodeveix, M. Filali, P. Farail, P. Gaufillet, H. Garavel, and F. Lang. FIACRE: an Intermediate Language for Model Verification in the topcased Environment. In *4th European Congress* Embedded Real Time Software *(ERTS)*, January 2008.

[5] B. Berthomieu, P-O. Ribet, and F. Vernadat. The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets. *International Journal of Production Research*, 42(14):2741–2756, July 2004.

[6] B. Combemale, L. Gonnord, and V. Rusu. A Generic Tool for Tracing Executions Back to a DSML's Operational Semantics. In *Modelling Foundations and Applications, 7th European Conference, ECMFA 2011, Birmingham United Kingdom.*

[7] S. Dal Zilio and N. Abid. Real-time Extensions for the Fiacre modeling language. In *MoVep 2010, Summer School on Modelling and Verifying Parallel Processes*, Aachen, Allemagne, June 2010. 6 pages FNRAE Quarteft.

[8] Z. Demirezen, M. Mernik, J. Gray, and B. Bryant. Verification of dsmls using graph transformation: a case study with alloy. In *Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*, MoDeVVa '09.

[9] H. Goldsby, Betty H. C. Cheng, S. Konrad, and S. Kamdoum. A visualization framework for the modeling and formal analysis of high assurance systems. In *Proceedings of the 9th international conference on Model Driven Engineering Languages and Systems*, MoDELS'06, pages 707–721, Berlin, Heidelberg, 2006.

[10] F. Jouault and I. Kurtev. Transforming Models with ATL. In *Satellite Events at the MoDELS 2005 Conference, Proceedings of the Model Transformations in Practice Workshop*, volume 3844 of *LNCS*, pages 128–138, Jamaica, 2005. Springer.

[11] J. Lilius and I.P. Paltor. vuml: a tool for verifying uml models. In *Automated Software Engineering, 1999. 14th IEEE International Conference on.*, pages 255 –258, oct 1999.

[12] P. Pelliccione, P. Inverardi, and H. Muccini. Charmy: A framework for designing and verifying architectural specifications. *IEEE Trans. Soft. Eng.*, 35(3):325–346, 2009.