Fiacre: an Intermediate Language for Model Verification in the Topcased Environment

Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, M Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang, François Vernadat

▶ To cite this version:

Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, M Filali, Hubert Garavel, et al.. Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. 4th European Congress ERTS Embedded Real Time Software (ERTS 2008), SEE : Société de l'électricité, de l'électronique et des technologies de l'information et de la communication; 3AF : Association Aéronautique et Astronautique de France, Jan 2008, Toulouse, France. pp.1-8. inria-00262442

HAL Id: inria-00262442
https://inria.hal.science/inria-00262442

Submitted on 11 Mar 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Fiacre: an Intermediate Language for Model Verification in the TOPCASED Environment

B. Berthomieu[1], JP. Bodeveix[2], P. Farail[3], M. Filali[2], H. Garavel[4], P. Gaufillet[3], F. Lang[4], F. Vernadat[1]

1: LAAS-CNRS, 7 avenue du Colonel Roche, F-31077 TOULOUSE
2: IRIT, Université Paul Sabatier-CNRS, 118 route de Narbonne, F-31062 TOULOUSE
3: AIRBUS France, 316 route de Bayonne, F-31060 TOULOUSE
4: INRIA Rhône-Alpes, 655 avenue de l'Europe, Montbonnot Saint-Martin, F-38334 SAINT ISMIER

Abstract: Fiacre was designed in the framework of the TOPCASED project dealing with model-driven engineering and gathering numerous partners, from both industry and academics. Therefore, Fiacre is designed both as the target language of model transformation engines from various models such as SDL, UML, AADL, and as the source language of compilers into the targeted verification toolboxes, namely CADP and Tina in the first step. In this paper, we present the Fiacre language. Then transformations from AADL to Fiacre are illustrated on a small example.

Keywords: model checking, model transformation, embedded systems, architecture description languages

1. Introduction

Fiacre[10] is an acronym for *Format Intermédiaire pour les Architectures de Composants Répartis Embarqués* (Intermediate Form for Architectures of Embedded Distributed Components). Fiacre is a formal intermediate model to represent both the behavioural and timing aspects of systems -in particular embedded and distributed systems- for formal verification and simulation purposes. Fiacre was designed in the framework of the TOPCASED project[11] dealing with model-driven engineering and gathering numerous partners, from both industry and academics. Therefore, Fiacre is designed both as the target language of model transformation engines from various models such as SDL, UML, AADL, and as the source language of compilers into the targeted verification toolboxes, namely CADP and Tina in the first step. Here, we focus on the front-end of this pipeline, namely on the Fiacre language and on the translation of a high level modelling language (AADL) to Fiacre. The rest of this paper is organized as follows: Section 2 presents the TOPCASED environment and its meta-level tools. Section 3 briefly introduces Fiacre as a pivot language for verifying high level modelling languages and describes in more details the translation of some aspects of AADL to Fiacre. Section 4 details the Fiacre language and its semantics. Section 5 illustrates parts of the translation of AADL to Fiacre on a communication protocol.

2. The TOPCASED environment

The TOPCASED project aims at developing an open source CASE tool for safety critical applications. It offers meta-level tools helping in the implementation of model driven environments which aim at designing models which can be verified and transformed into executable code.

The TOPCASED environment is based on the Eclipse platform. It is organized around a model bus allowing the communication between editing, verification, simulation and code generation tools. It offers meta-level tools to help in the implementation of dedicated CASE tools:

- Graphic EMF metamodel editors
- Static semantic verification or analysis tools based on the OCL[7].
- Tree-based and graphic model editors can be generated from an EMF metamodel.
- Model transformation languages (ATL[4], Kermeta[3]) and code generation languages (OAW[5], Acceleo[6]) are provided to support model driven development.

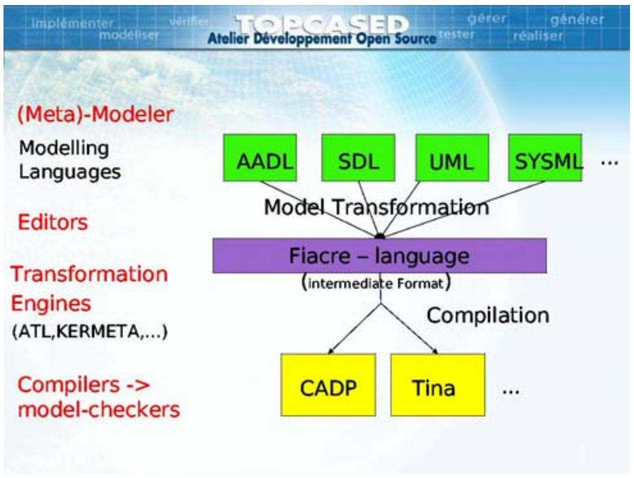
Metamodels and editors are provided by TOPCASED for well known modelling languages, such as UML, SysML and AADL.

3. Verification in TOPCASED

Model verification is one of the most important objectives of TOPCASED. A support for the static analysis of models already exists through an OCL interpreter. OCL can be used to specify the static semantics (type checking conditions, usage restrictions, ...) of modelling languages and marginally to specify dynamic properties. Here, we focus on tools allowing the verification of dynamic properties of real time systems. This paragraph presents the architecture of the verification framework and presents in more details the various elements that interact in the verification process.

3.1 Architecture of the verification framework

The modelling language of model checkers must be kept simple enough for the tool to be efficient. It is thus preferable to reuse existing model checkers (Tina and CADP) to check properties of high level languages (UML, SysML, AADL), which leads to the introduction of a transformation phase between the two levels. In order to make easer the connection of model checkers to those languages, we have introduced the pivot language Fiacre so that the expression of the run-time semantics of high level languages can be factored and expressed using the pivot. It must be powerful enough to support the expression of the semantics of real time preemptible systems even if back-end model checkers do not take into account all the aspects of the model. The TOPCASED environment is exploited to implement this process, as shown by the following figure:



TOPCASED offers the metamodel of the high level modelling languages together with graphical editors. The abstract syntax of the Fiacre intermediate language is defined through its metamodel. Transformation languages are used to generate Fiacre models from modelling languages. At this step, some semantics choice must be performed to identify relevant subsets of sources languages and reduce the complexity of intermediate models. Then, source code generators are used to produce the text representation of the Fiacre model and communicate with the external Fiacre front-end. The Fiacre tool performs static analysis of its entry and generates Tina and CADP models which are analysed by the corresponding tools.

It has to be noted that this schema does not illustrate the reverse path which should take as inputs the counter-examples produced by CADP or TINA, convert them to a (to be defined) Fiacre trace model and finally to trace models of each high level modelling language. Such traces should be played back using simulation tools.

3.2 Verification of high level modelling languages

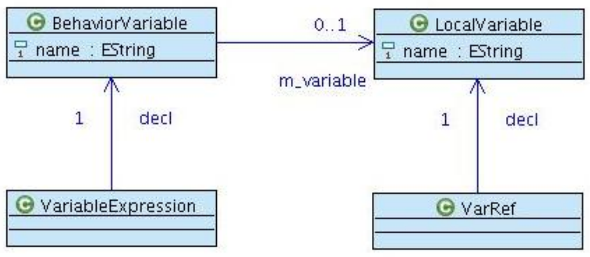
High level modelling languages have a very complex semantics and only subsets of them will be taken into account. Uses of these languages must be defined, which correspond to defining profiles in the UML terminology. For example, AADL can be used as a synchronous language. The validity of the synchronous hypothesis can be checked independently. Then, translating AADL models to a non-temporised model checker is sufficient. AADL can also be used as a totally asynchronous language with preemptive scheduling. In this case, even reachability analysis becomes undecidable, but the semi-decision algorithm provided by Tina can be used. Intermediate choices can be performed as for example ignoring preemption, considering task have fixed duration, ignoring remote procedure calls, ignoring concurrent accesses to shared variables, ...

3.3 Translating AADL to Fiacre

We illustrate this framework with the translation of AADL models to Fiacre. This transformation is performed using Kermeta. Kermeta is a model transformation language which combines features coming from Eiffel, Java, OCL and aspect oriented programming and offers easy access to EMF model repositories. Two main features are heavily used in the translation: high level iterators on lists and aspect oriented annotations. These annotations allow to specify extensions of existing classes with new attributes and operations. The attributes are used to memorize the Fiacre objects resulting from the transformation of the AADL objects. Otherwise, external data structures such as hash tables should be used.

3.3.1 Aspect Oriented Programming

Aspect oriented programming is illustrated through the translation of variables declared in the AADL behavioural annex to Fiacre variables. For each occurrence of a variable in an expression, a Fiacre variable must be created and contains a reference to its declaration, which is the image through the transformation process of the declaration point of the annex variable. This structure is described by the following class diagram:



In this schema, classes on the left side belong to the source metamodel while classes on the right side belong to the target metamodel. An important property to be ensured by the transformation is that the diagram commutes: the VarRef image of a VariableExpression refers to the LocalVariable which is the image of BehaviorVariable referenced by the source VariableExpression.

This property is established by adding the m_variable reference to the BehaviorVariable metaclass thanks to Kermeta aspect oriented features:

```
@aspect "true"
class BehaviorVariable {
reference m_variable: Fiacre::LocalVariable
}
```

The following code describes the transformation of BehaviorVariable instances. A new image is created only if this has not been done yet `(flv=void)`.

```
operation transformVariable
   (bv: behavior::BehaviorVariable):
     Fiacre::LocalVariable is do
  var flv:Fiacre::LocalVariable
  flv := bv.m_variable
  if (flv==void) then
    flv:= Fiacre::LocalVariable.new
    flv.name:=bv.name
    if component::DataClassifier.isInstance
       (bv.type) then
      var dc: component::DataClassifier
      dc?=bv.type
      flv.type := data.transformData(dc)
    else
      -- error
    end
    bv.m_variable := flv
  end
  result:=flv
end
```

3.3.2 Transformation of an AADL architecture

An AADL system is specified by a hierarchical structure of software and hardware components. Here, we ignore hardware components so that the system can be seen as a hierarchy of components, the leaves of which are threads encapsulating behaviours. Threads communicate asynchronously through ports and shared variables, and synchronously using remote procedure calls. We only consider here asynchronous port communication, which differs from Fiacre synchronous communication. In order to offer to threads a simple access to their ports, we introduce at each level of the hierarchy a new component called a glue which desynchronizes sender and receiver ports and makes received data accessible to consumers at the moment specified by the AADL semantics: data is provided to a thread when it is dispatched. Given the considered restrictions, it cannot receive new data while it is active. Data is either stored in a one place buffer or queued. Queued data is made totally or partially visible to the thread, depending on port properties. A flag indicates if new data has arrived since the previous dispatch. The size of the visible part of the queue can also be accessed by the thread. Thus, an AADL thread has access to its environment via the glue through one or two channels for each AADL port: a data channel and possibly a control channel.

3.3.3 Transformation of thread behaviours

Thread behaviours are expressed using the behavioural annex we have proposed for AADL[2]. A thread behaviour is mainly an automaton, the transitions of which are triggered by incoming events and perform local state changes and communications. As previously explained, communications are controlled by the AADL semantics and pass through the glue process. So, a thread is translated into a Fiacre process taking as parameters ports linked to the glue. Annex specific constructs associated to the AADL programming interface such as checking the fresh status of incoming data or the number of messages in input event (data) queues are translated to communications through control channels managed by the glue.
Thread dispatch is another important aspect of AADL execution model. It is not yet fully managed as temporised aspects are not considered, but dispatch is synchronized with data transfers within the glue: data seen by dispatched processes is that of the time of dispatch.

3.4 The Tina verification environment

TINA is a software environment to edit and analyze Petri nets, Time Petri nets, and Time Transition Systems, and extension of these nets handling data, priorities and temporal preemption. Beside the usual editing and analysis facilities of similar environments, the essential components of the toolbox are a state space abstraction tool (also called tina) and a model checking tool (selt). More details about the toolbox capabilities can be found in [8].

TINA offers various abstract state space constructions that preserve specific classes of properties of the state spaces of nets, like absence of deadlocks, linear time temporal properties, or bisimilarity. For untimed systems, abstract state spaces help to prevent combinatorial explosion. For timed systems, TINA provides various abstractions based on state classes, preserving reachability properties, linear properties or branching properties. Mode details about these constructions can be found in [15].

State space abstractions are provided in various formats suitable for existing model checkers. The TINA toolbox also provides a native model checker, selt. Selt allows one to check more specific properties than the general ones (boundedness, deadlocks, liveness) already checked by the state space generation tool. Selt implements an extension of linear time temporal logic known as State/Event LTL [SELTL], a logic supporting both state and transition properties. The modelling framework consists of Kripke transition systems (labelled Kripke structures, the state class graph in our case), which are directed graphs in which states are labelled with atomic propositions and transitions are labelled with actions.

State/Event-LTL formulas are interpreted over the computation paths of the model. They may express a wide range of state and/or transition properties, some typical formulas are the following (a formula evaluates to true if it does so on all computation paths, **X**, **F, G** and **U** are LTL modalities, *p, q* are formulas):

p	*p* holds at the start
X *p*	*p* holds at the next step (next)
G *p*	*p* holds all along the path (globally)
F *p*	*p* holds in a future step (eventually)
p **U** *q*	*p* holds until *q* holds (until)

Realtime properties, like those expressed in so-called "timed temporal logics", are checked using the standard technique of observers, encoding such properties into reachability properties. The technique is applicable to a large class of realtime properties and can be used to analyse most of the ``timeliness'' requirements found in practice.

3.5 The CADP verification environment

CADP (Construction and Analysis of Distributed Processes), is a toolbox for the design of communication protocols and distributed systems, connected to various complementary tools.

CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking, including:

- Compilers for several input formalisms: High-level protocol descriptions written in the ISO language LOTOS, Low-level protocol descriptions specified as finite state machines, and Networks of finite state machines;
- Equivalence checking tools for e.g. minimization and comparisons modulo bisimulation relations;
- Model-checkers for various temporal logic and mu-calculus;
- Several verification algorithms for enumerative, on-the-fly and symbolic verification, compositional minimization, partial orders, and distributed model checking;
- A number of other tools with advanced functionalities such as visual checking, performance evaluation, etc.

CADP is designed in a modular way and puts the emphasis on intermediate formats and programming interfaces (such as the BCG and OPEN/CAESAR software environments), which allow the CADP tools to be combined with other tools and adapted to various specification languages. A recent description of the toolbox can be found in [9]

4. The Fiacre language

This paragraph presents the main characteristics of the Fiacre language, how Fiacre programs are verified and a small example.

4.1 Introduction to Fiacre

Fiacre was developed as an intermediate formal language for the Topcased project to represent both the behavioural and timing aspects of systems — in particular embedded and distributed systems — for formal verification and simulation purposes.

The language, formally described in [8], embeds the following notions:

- *Processes* describe the behaviour of sequential components. A process is defined by a set of control states, each associated with a piece of program built from deterministic constructs available in classical programming languages (assignments, if-then-else conditionals, while loops, and sequential compositions), non-deterministic constructs (non-deterministic choice and non-deterministic assignments), communication events on ports, and jumps to next state.
- *Components* describe the composition of processes, possibly in a hierarchical manner. A component is defined as a parallel composition of components and/or

processes communicating through ports and shared variables. The notion of component also allows to restrict the access mode and visibility of shared variables and ports, to associate timing constraints with communications, and to define priorities between communication events.

Fiacre is designed both as the target language of model transformation engines from various models used with the TOPCASED project such as SDL or AADL, and as the source language of compilers into the targeted verification toolboxes, namely CADP [9] and Tina [8] in the first step.

Fiacre was directly inspired from two works, namely V-Cotre [18] and Ntif [19], and indirectly by a number of research works in concurrency theory and real-time systems theory over the last two decades. Its timing primitives are borrowed from Time Petri nets. Time constraints and priorities have been integrated so that the properties of components can be deduced from the properties of their constituents (compositionality property), as in the BIP framework [16]. Concerning compositions, Fiacre makes use of a “graphical” parallel composition operator previously found in E-Lotos [17].

4.2 Verification of Fiacre programs

For verification purposes, Fiacre programs are compiled into suitable input formalisms for the Tina and CADP toolboxes. The Fiacre compilers for both toolboxes share their front-end (syntactical analysis and type-checking). The Tina back-end translates Fiacre programs into Time Transition Systems (an extension of Time Petri nets handling data and priorities) and the specification requirements into State-Event LTL formulas. The CADP back-end translates Fiacre programs into LOTOS programs, which are then handled by the dedicated LOTOS tools embedded into the CADP toolbox (translation into C code for simulation, testing and verification).

4.3 A transmission protocol in Fiacre

The following code specifies a timed version of the alternated bit protocol. It contains the two communicating processes (sender and receiver) and the unsafe communication media (mmedium and amedium). Timings are specified in the encapsulating abp component. For example, the timeout channel is declared with the timing interval [5,6]. It means that synchronization through this channel must be performed after 5 time units have elapsed in the wait state of the sender, and before 6 time units.

```
type data is ... (* left unspecified *)
channel msg is bool # data

process sender [inp: data, outp: msg,
     timeout, ack: none] (ssn: bool) is
  states idle, send, wait, resend
  var data: data
  from idle inp? data; to send
  from send
    outp! ssn,data; ssn := not ssn; to wait
  from wait
    select ack; to idle
     []    timeout; to resend
    end
  from resend outp! ssn,data; to wait

process receiver [inp: msg, outp: data,
     ack: none] (expected:bool) is
  states idle, accept, ack
  var data:data, ssn:bool
  from idle
    inp? ssn,data;
    if ssn = expected then to accept
    else to ack end
  from accept
    ssn := not ssn; outp! data; to ack
  from ack ack; to idle

process mmedium [get: msg, lost: none,
     put: msg] is
  states get; put
  var b:bool, d:data
  from get get?b,d; to put
  from put
    select put!b,d; to get
    []     lost, to get
    end

process amedium [get: none, lost: none,
     put: none] is ...
 (similar to mmedium except types of ports)

component abp [inp: data, outp: data] is
  port timeout in [5,6],
       min, mout: msg in [0,1],
       ain, aout: none in [0,1],
       mloss, aloss: none in [0,1]
  par sender [inp,min,timeout,aout](false)
  ||  mmedium [min, mloss, mout]
  ||  receiver [mout,outp,ain](false)
  ||  amedium [ain, aloss, aout]
  end
```

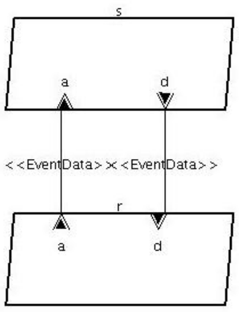
Remarks

- The use of time intervals associated to ports can be considered as a design decision resulting from the Cotre project[20]. Actually, we have considered this feature as more suitable than the one which consists in using clocks explicitely. In fact, this solution introduces the usual timers as special ports. It follows that times are automatically, and consequently safely, set and reset.
- In this paper we have not talked about timing issues related to process scheduling. More precisely, we have not considered how processes should be specified as periodic, sporadic, ... Again, some solutions to this problem have already been experimented

within the Cotre project. We are currently experimenting alternative solutions. Our goal is to be able to specify generic solutions to real-time scheduling problems.

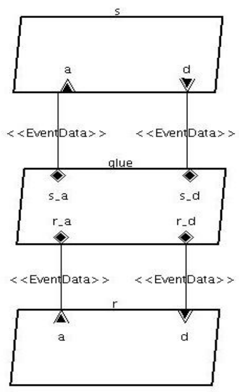
5. An illustrative transformation example

In this section, we illustrate the translation process from a high level description in AADL to a Fiacre description. We sketch such a translation using the Kermeta transformation language. We take as an example the alternating bit protocol. It is described in AADL by a system implemented as the composition of two processes, s instance of the sender process type, and r, instance of the receiver process type. They communicate through two asynchronous unidirectional communication channels which are supposed unsafe. Data and a status bit are exchanged through ports named d while acknowledgements use ports named a. The AADL architecture of the protocol is illustrated by the following figure and described by the AADL system implementation abp.i.



```
system implementation abp.i
subcomponents
  s: process sender.i;
  r: process receiver.i;
connections
  event data port s.d -> r.d;
  event data port r.a -> s.a;
end abp.i;
```

The Fiacre translation of the architecture introduces a glue process which manages asynchronous communications using synchronous communications between the sender root_s and the glue, and the receiver root_r and the glue. The new architecture is illustrated by the following AADL model:



In Fiacre, the processes root_r and root_s are composed using interleaving. Their product is synchronized with the glue.

```
component root is
port
  root_s_dGlue : in out root_s_d_ty,
  root_r_dGlue : in out root_r_d_ty,
  root_r_aGlue : in out root_r_a_ty,
  root_s_aGlue : in out root_s_a_ty
par
  root_s_dGlue,root_r_dGlue,root_r_aGlue,
    root_s_aGlue ->
  par
      -> root_s[root_s_dGlue, root_s_aGlue]
   || -> root_r[root_r_dGlue, root_r_aGlue]
  end
||
  root_s_dGlue,root_r_dGlue,root_r_aGlue,
    root_s_aGlue ->
  rootGLUE[root_s_dGlue, root_r_dGlue,
           root_r_aGlue, root_s_aGlue]
end
```

Now, we consider the implementation of the receiver. AADL does not support the specification of detailed behaviours. It is either provided through a link to the actual code or specified in extensions of AADL called annexes. Here, we use the behavioural annex we have proposed [2]. The receiver code is described as a two states automaton which waits for data. An acknowledge is returned and the receiver changes its state if it receives the expected data.

```
thread implementation t_receiver.i
subcomponents
  v: data V.i;
annex behavior_specification {**
states
  st: initial complete state;
  sf: complete state;
transitions
  st -[d?(v) when v.b]-> sf {a!(v.b);};
  st -[d?(v) when not v.b]-> st {a!(v.b);};
  sf -[d?(v) when not v.b]-> st {a!(v.b);};
  sf -[d?(v) when v.b]-> sf { a!(v.b); };
**};
end t_receiver.i;
```

This AADL thread is translated into the following Fiacre process which takes as parameters the channels linked to the glue. The translation is here straightforward.

```
process root_r_t[d: in V_i,
                 a: out boolean] is
  states st, sf
  var v : V_i
  init to st
  from st d?v where v.b; a!v.b; to sf
  from st d?v where not v.b; a!v.b; to st
  from sf d?v where not v.b; a!v.b; to st
  from sf d?v where v.b; a!v.b; to sf
```

Timeouts should be added to the AADL model to conform with the actual protocol. In fact, they can be expressed both in the source (behavioural annex) language and in the target (Fiacre) language but are not yet supported by the translator. For example, the following transition should be added to the AADL model:

```
st -[on timeout T]-> st {a!(false);};
```

It sends again a negative acknowledgement if the next message is not received with T time units. This transition informs the scheduler to dispatch the thread when a message arrives or after the specified timeout.

It should be possible to verify some LTL properties such as the following: if each kind of messages crosses the glue infinitely often, the receiver reaches each of its two states infinitely often.

6. Conclusion and Perspectives

In this paper, we have presented the current state of the development of the Fiacre intermediate language for model verification in the TOPCASED environment. It should be stressed that many design decisions have been made based on previous research projects and especially the Cotre project[20]. With respect to model checking, Fiacre relies on mature tools like CADP and Tina. However, we have to keep in mind that the combinatoric explosion hinders the verification process. We have to look for abstraction mechanisms which enhance it. Some research directions have been suggested. Of course, these abstractions should be made concrete for code generation. The TOPCASED environment, through its transformation engines, should make easy such a task.

Status of implementations

The metamodel of the source and target languages are available: AADL metamodel is provided by the OSATE[21] tool integrated in TOPCASED and the Fiacre metamodel has been build using the TOPCASED environment. The implementation of the AADL to Fiacre translator is partial for the moment. It supports the translation of architectural and behavioural aspects. Work now focuses on the specification in Fiacre of scheduling and temporised behaviours. These aspects will have a great impact on the overall performance of the analysis tool. So subsets of AADL will have to be specified in order to avoid producing too complex models. They range from fully synchronous models to fully asynchronous ones with preemptive scheduling. Management of time is also important. It can be either completely abstracted, considered as discrete or dense. Fiacre and the Tina model checker will offer a support for all these variants.

Perspectives

In this paper, we have discussed about the basic Fiacre. Currently, we are also investigating extensions that could be interesting in order to make easier the translation from high level description languages or to make more efficient the verification process. Among them, let us mention:

- The notion of modes [14]: actually such a feature is currently available in some architecture description languages, e.g. Giotto, AADL. It would be interesting to offer such a notion with a dedicated verification process.
- The generalization of the send-receive primitives. For instance, one can envision a statement where multiple sends and receives could be done atomically in one step. Then, such a step could be analyzed as a whole and translated afterwards into basic send and receive statements.
- The verification of parameterized systems: currently, a system is composed of a fixed number processes and components. Each system of a same family must be verified separately. Fiacre could be enriched to specify families of systems with regular architectures, e.g., a bus, a ring, ... It follows that dedicated verification techniques could be used in order to go beyond the traditional model checking techniques where, for instance, the number of components, the size of buffers, ... have to be fixed a priori.

7. References

[1] SAE Aerospace: "Architecture Analysis & Design Language (AADL)", AS-5506, SAE International, 2004.

[2] Ricardo Bedin Franca, Jean-Paul Bodeveix , David Chemouil, Mamoun Filali, Dave Thomas, Jean-François Rolland: "The AADL behaviour annex experiments and roadmap", UML&AADL'2007, (Auckland, New Zealand), 2007.

[3] Pierre-Alain Muller, Franck Fleurey, Didier Vojtisek, Zoé Drey, Damien Pollet, Frédéric Fondement, Philippe Studer, and Jean-Marc Jézéquel. « On executable meta-languages applied to model transformations » In *Model Transformations In Practice Workshop*, Montego Bay, Jamaica, October 2005.

[4] Jouault, F, and Kurtev, I : Transforming Models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica.

[5] OAW, http://www.openarchitectureware.org/

[6] Acceleo, http://www.acceleo.org/

[7] OCL, UML 2.0 Object Constraint Language, http://www.omg.org/cgi-bin/doc?ptc/2003-10-14.pdf

[8] B. Berthomieu, P.-O. Ribet, F. Vernadat, The tool TINA -- Construction of Abstract State Spaces for Petri Nets and Time Petri Nets, International Journal of Production Research, Vol. 42, No 14, July 2004.

[9] H. Garavel, F. Lang, R. Mateescu, W. Serve. « CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes », 19th Int. Conf. On Computer Aided Verification (CAV'07), July 2007.

[10] B.Berthomieu, J.P.Bodeveix, M.Filali, H.Garavel, F.Lang, F.Peres, R.Saad, J.Stoecker, F.Vernadat. "The syntax and semantics of Fiacre ». Rapport LAAS N°07264, Rapport de Contrat, Projet ANR05RNTL03101 OpenEmbeDD, Mai 2007, 30p.

[11] Topcased: "Toolkit in OPen-source for Critical Apllications and SystEms Development", http://www.topcased.org

[12] D. Garlan and R. Monroe and D. Wile: "ACME: An Architecture Description Interchange Language", CASCON'97, Toronto, 1997.

[13] Robert Allen: "A Formal Approach to Software Architecture", Carnegie Mellon, School of Computer Science, 1997.

[14] Jean-François Rolland, Jean-Paul Bodeveix, Mamoun Filali, Dave Thomas and David Chemouil: "Modes in asynchronous systems", UML&AADL'08, IEEE ICCECS, Belfast (to appear).

[15] B. Berthomieu, F. Vernadat, State Space Abstractions for Time Petri Nets, Handbook of Real-Time and Embedded Systems, Ed. I. Lee, J. Y-T Leung, S. H. Son, Chapman & Hall/CRC, 2007.

[16] A. Basu, M. Bozga, J. Sifakis. « Modeling heterogeneous real-time systems in BIP ». 4th IEEE Int.l Conf. on Software Engineering and Formal Methods (SEFM06), Pune, Sept 2006.

[17] ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization --- Information Technology, Genève, Sept 2001.

[18] B. Berthomieu, et al. « Towards the verification of real-time systems in avionics: the Cotre approach. Proceedings of the 8th Int. Workshop on Formal Methods for Industrial Critical Systems, 2003.

[19] H. Garavel and F. Lang. « NTIF: A general symbolic model for communicating sequential processes with data ». 22nd IFIP WG 6.1 Int. Conference on Formal Techniques for Networked and Distributed Systems, 2002.

[20] J.-M. Farines, B. Berthomieu, J.-P. Bodeveix, P. Dissaux, P. Farail, M. Filali, P. Gaufillet,H. Hafidi, J.-L. Lambert, P.Michel, F. Vernadat. « The Cotre Project: Rigorous Software Development for Real Time Systems in Avionics ». Electronic Notes in Theoretical Computer Science. http://www.sciencedirect.com/science/journal/15710661 Volume 80. August 2003, Pages 203-218. Eighth International Workshop on Formal Methods for Industrial Critical Systems (FMICS'03).

[21] The SEI AADL Team: "An Extensible Open Source AADL Tool Environment (OSATE)",Software Engineering Institute, 2006.

8. Glossary

AADL	Architecture Analysis & Design Language
ADL	Architecture Design Language
ATL	Atlas Transformation Language
CMU	Carnegie Mellon University
EMF	Eclipse modeling Framework
OSATE	Open Source AADL Tool Environment
TOPCASED	Toolkit in Open Source for Critical Applications & Systems Development
UML	Unified Modeling Language
SEI	Software Engineering Institute
XMI	XML Metadata Interchange