Behavioral Verification of UML2.0 Software Architecture

Sakka Rouis Taoufik[#1], Bhiri Mohamed Tahar[*2], Kmimech Mourad[**3]

[#] Crystal Laboratory, ENSI, University of Manouba
2010 Manouba, Tunisia
[1]srtaoufik@yahoo.fr

[*] Miracl Laboratory, ISIMS, Technological Pole of Sfax
BP 242 - 3021, Sakiet Ezzit Sfax, Tunisia
[2]tahar_bhiri@yahoo.fr

[**] UR-OASIS Laboratory, ENIT, University of Tunis El Manar
1002 Tunis,Tunisia
[3]mkmimech@gmail.com

Abstract—**The architecture description languages (ADL) have been developed to formally model software architectures. One of the challenges of ADLs is their ability to perform the validation and/or verification of software system properties very early in their life cycle and throughout their development process. The Wright formal ADL is based on CSP and, thus, inherits its equivalence and refinement relations as well as support for the verification of general properties using the FDR2 model checker. In this paper, we proposed to open UML2.0 on the Wright ADL in order to verify the behavioral consistency of UML2.0 software architectures. To achieve this, we suggested to translate this software architecture into a Wright specification. Using Wr2fdr tool, these Wright descriptions were automatically translated to a CSP specification acceptable by the FDR2 model-checker.**

Keywords—ADL; Verification; Software Architecture; Behavior; UML2.0; Model-Checker; FDR2.

I. INTRODUCTION

Today, the architecture description languages (ADL) are being developed to formally model software architectures. Each ADL is focused on a particular architecture family and offers a modeling language and technical analysis which are highly specialized. For example, Acme an exchange language between ADLs and Rapide is focused on the specification and analysis of the dynamic aspect of architectures. Most of the tools supporting ADLs offer a structural analyzer of architectural elements. However, rarely do these tools propose a behavioral analysis of these architectural elements. This amounts to the ambiguity of what an ADL should describe at the behavioral level.

The UML 2.0 standard provides several models for the specification of the structural and behavioral aspect of architectural elements. Indeed, the UML 2.0 component model is especially well suited to depict the structural systems based on components. To describe the behavior view of a component or of its interfaces or ports, UML2.0 proposes four behavior specification mechanisms (subtypes of behavior) to provide concrete semantics of the generic behavior model; the subclasses provided are Interaction, Activity, State Machine (SM) and its specialization Protocol State Machine (PSM). These SMs are used to describe the component behavior as part of its implementation, while the PSMs define the protocol interfaces which can be considered as a specialization of the UML state machines without action or activity. The PSM can describe only one direction of communication. It also means that PSM cannot describe the communication relationships between the requested and provided interfaces.

However, none of the tools supporting UML 2.0 offers behavioral analyzers. This gives birth to works that open UML on formal languages to verify the behavioral consistency of UML diagrams. For example, the work described in [2] proposes to translate UML2.0 state machines towards an LTS specification verifiable by LOTOS tool. In [22], the authors propose to formalize a Protocol State Machine (PSM) in Alloy lightweight formal modeling language. Using Alloy analyser, this formalization can be used to simulate and verify consistency between UML artefacts such as components. The formalization of PSMs is implemented using the model transformation language ATL [3]. The authors of [1] propose to translate the UML sequence diagrams to a CSP formal specification verifiable by the FDR2 [5] model-checker.

In this study, we target the formal verification of software architectures described by UML2.0 component assemblies endowed with a behavioral specification described by Port State Machines (PoSM) [4]. Where PoSM is an extension of UML2.0 Protocol State Machine (PSM). The principal contributions of this PoSM model are the ability to define the required and provided behavior and the possibility to differentiate between a request and a response. An operation call of a component which is inherently non-atomic is modeled in PoSM with two atomic events: a request (corresponding to the start of the call) and a response (corresponding to its termination). To achieve this, the model transformation approach is used to translate the UML2.0 architectural elements into Wright specification. This formal ADL is opted for as an intermediate modeling language. The choice of this ADL is justified by the existence of the Wr2fdr tool. The latter can translate a Wright specification into another in CSP acceptable by the FDR2 model-checker. The Wr2fdr tool generates a set of contracts aimed at connecting behavioral compatibility and compatibility between the interfaces of a component and the component itself.

978-1-5090-4795-6/16 $31.00 © 2016 IEEE
DOI 10.1109/SKG.2016.29

This article is structured as follows: Section 2 presents the main related work on behavioral verification in ADLs. Section 3 proposes an overview of our behavioral verification approach. Section 4 deals with our systematic rules allowing the translation of UML2.0/PoSM source model to the Wright/CSP target model. Section 5 exhibits a validation of our approach on an ATM/Bank system. Finally, Section 6 provides a conclusion and possible future work.

II. Behavioral analysis in ADLs

Most ADLs share the same architectural concepts such as components, connectors and configuration. One of the challenges of ADLs is their ability to perform the validation and/or verification of software system properties very early in their life cycle and throughout their development process. However, few ADLS allow an explicit description of the behavioral property. That amounts to the ambiguity of what an ADL should describe at the behavioral level. In this section, the main ADLs supporting this behavioral property are presented.

Pi-ADL [6] is a formal language with a solid theoretical foundation. It was designed as part of the European project ArchWare [7]. This language is a well-trained extension to Pi-Calcul [8] aimed at the structural and behavioral descriptions of software architectures based on components. The behavioral aspect of a Pi-ADL architectural element is expressed using scheduled actions. As Pi-Calcul is complete in the Turing sense, any behavioral aspect can be modeled using Pi-ADL. Additionally, Pi-ADL offers structural and behavioral properties. The latter are expressed in the AAL ArchWare analysis language [9] which is able to verify these properties. The authors of [10] propose to use the Input Output Symbolic Transition System (IOSTS) for the verification of the structural and behavioral implementation conformity with its architecture described in Pi_ADL.

The Tracta project [11] proposes an extension of Darwin ADL which allows the addition of a behavioral specification of primitive components and a set of properties within an architectural description. This behavioral information is described at a high level language derived from CSP [23] called FSP [12]. The LTSA analysis tool detects the presence of a deadlock and checks these FSP properties.

Although Fractal [13] does not provide a mechanism for specifying the behavioral aspects of architectural elements, various ongoing works come to extend this model through different contract levels. For example, ConFract [14] is working on defining assertions at the Fractal component interfaces. The authors of [15] propose to use the LTS system to specify the behavioral aspects of Fractal components.

The AADL [16] international standard provides two fundamental mechanisms (property and annex) that can be used for specifying any notion according to the needs of the user. In [17], these mechanisms are used for specifying the components behavior, but there are no semantics defined in the standard to detect problems with this behavior. The work presented in [18] proposes to open AADL on Petri nets to verify certain behavioral properties. The main weakness of AADL is the lack of rigorous operational semantics. The authors of [19] deal with this problem by providing a translation of AADL to BIP, which has formal operational semantics defined in terms of labeled transition systems (LTS). This translation allows the analysis and detection of potential deadlocks and the verification of different properties.

The UML2.0 standard provides a number of diagrams capable of describing the structural and behavioral aspects of component-based software architecture. For example, the UML 2.0 component model provides the basic architectural concepts such as: component, connector, port, interface, etc. The introduction of these concepts and the distinction between available and required interfaces provide a range of interesting elements for the modeling of component-based systems. To describe the behavioral communication of interconnected component interfaces, UML2.0 suggests a new model called Protocol State Machine (PSM). The latter is a specialization of UML State Machine (SM) without action or activity. The PSM can describe only one communication direction. It also means that PSM cannot describe the communication relationships between the requested and provided interfaces. Thus, the profile position Port State Machine (PoSM) is selected. This is considered as an extension to the PSMs. The main contributions of this model are the ability to define the required and provided behaviors and the possibility to differentiate between a request and a response. An operation call of a component which is inherently non-atomic is modeled in PoSM with two atomic events: a request (corresponding to the start of the call) and a response (corresponding to its termination). Moreover, in order to capture the interleaving of sent and received operation calls (via different interfaces of a Port), PoSMs explicitly distinguish between sent and received events (calls). To hide such technical details, PoSM employs these prefixes (! Or ?) and suffixes (↑ or ↓) to express the behavioral aspect. These shortcuts are inspired by the notations used in Behavior Protocols (BP) [20].

There, the event ?E stands for receiving an event E and !E for sending an event E. A call of an operation Oper is captured with a pair of atomic events; in the event labels, the suffix ↑ denotes a request and ↓ a response. For example, sequence ?Oper↑ ; !Oper↓ (here ; is the operator for sequencing) models receiving call of the operation Oper as receiving a request for Oper and sending a response.

The Wright ADL [21] provides the basic architectural concepts such as: component, connector, configuration and style. In Wright, a component (respectively a connector) may be provided with one or more interfaces called ports (roles respectively). Wright was one of the first approaches to allow the description of the behavior of architectural elements. The behavior of a Wright component (respectively of a connector) is described locally through the ports (respectively roles) and generally through a computation (glue respectively) using a type of CSP process algebra [23]. In addition, Wright defines eleven standard properties related to the consistency of software architecture, among which four -assimilated to behavioral contracts- are automated by the Wr2fdr tool. These four behavioral properties are specified informally as follows:

- Port/Computation consistency: the port specification should be a Computation projection, under the condition that the environment obeys the specification of all other ports. Intuitively, Property 1 states that the component does not care about events not covered by the ports.
- Connector without deadlock: The glue of a connector interacting with roles must be without deadlock. The connector description must verify that the roles coordination by the glue is consistent with the expected component behavior, knowing that a CSP process is said to be in deadlock situation when it can opt out at any event, but has not so far terminated correctly (by participating in the event §). Conversely, a process is without deadlock if it can never be in deadlock situation.
- Role without deadlock: each role of a connector must be without deadlock. Another category of inconsistency is detectable as a deadlock situation, when a role specification is itself inconsistent.
- Compatibility port / role: any port attached to a role must always continue its protocol in a direction that the role can have.

These properties are implemented by the Wr2fdr tool as a set of contracts aimed at connecting behavioral compatibility and compatibility between the interfaces of a component and the component itself. These contracts can automatically be checked by the FDR2 model checker FDR2.

III. Verification Approach

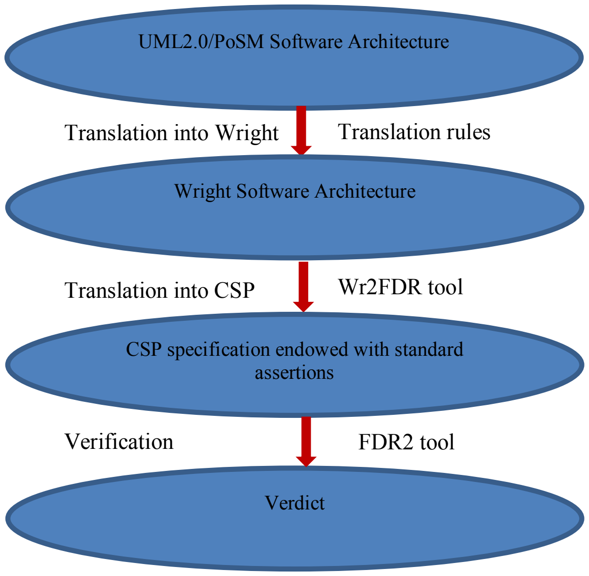

UML2.0/PoSM Software Architecture
Translation into Wright
Translation rules
Wright Software Architecture
Translation into CSP
Wr2FDR tool
CSP specification endowed with standard assertions
Verification
FDR2 tool
Verdict

Figure 1. Verification approach

Our verification approach presented in Figure 1 can be considered as three consecutive steps. Firstly, we propose to use the UML2.0 component diagram to describe the structural aspect of software architecture, and the PoSM model (see Section 2) to describe the behavioral aspect of this UML2.0 software architecture. Secondly, the translation rules of UML2.0/PoSM software architecture into Wright description are proposed. Finally, the Wright description obtained is translated into a CSP specification through the Wr2fdr tool which can automatically be checked by the FDR2 model checker.

IV. Translation Rules

In this section, we propose a set of rules allowing translating an UML2.0 / PoSM components assembly into Wright/CSP specification.

A. *Components translation*

Regarding the static aspect, we considered to translate an UML2.0 component by a Wright component according to the following rules:

- The name of this Wright component is that of the UML2.0 component.
- Any port associated with a UML 2.0 component is translated by a Wright port with the same name.

For the formalization of the behavioral aspect of an UML2.0 component in Wright/CSP, we proposed to translate those PoSM state machines by the CSP process. This systematic formalization is inspired by the works [24] and [1] binding, respectively, the state machine diagrams and the sequence diagrams to CSP. Specifically, we have offered to translate any state machine modeling the behavior of an UML2.0 port with a CSP process modeling the behavior of the machine which models the Wright port corresponding to the Wright component. Similarly, the state overall behavior of an UML2.0 component is translated into a CSP process modeling the computation behavior of the corresponding Wright component.

A PoSM transition is a simple UML transition that models the events invoking operations on its own from other components. In a PoSM, each event is prefixed by one of the operators (! or ?) and suffixed by one of the operators (↑ or ↓). There, ?Oper stands for receiving observed event Oper and !Oper for sending initialized event Oper. A call of an operation Oper is captured with two atomic events, where the event's name Oper has either the suffix ↑ for request or ↓ for response.

It can be concluded that for each operation call, four types of events can be present in a PoSM state machine. These types are: sending a request to execute an operation, receiving the request, sending a response and receiving the response. For this, four rules to translate a simple transition PoSM by a CSP event were proposed. The translation rules are the following:

- Sending a request to execute an operation, translated by an initialized event having the same name as this operation.
- Receiving a request to execute an operation, translated by an observed event having the same name as this operation.
- Sending a response to an execution of an operation, translated by an initialized event. The name of this event is that of the operation preceded by the R character (R to say Response).

- Receiving a response to an execution of an operation, translated by an observed event. The name of this event is that of the operation preceded by the R character (R to say Response).

A transition to a final state is translated by the successfully terminated event: TICK (noted in CSP by √).

Delayed transitions by a compound state are translated with a Pcomp CSP process. The latter is composite with Pci sub_process separated by one of the following operators: |~| or []. The formalization of this composition process is presented by the following formula:

Pcomp = $\sum_{i=1}^{n}$ operator Pci where n is the number of transitions and operator is:

- [] : if the choice between these transitions is an external choice. In other words, if these Pci are observed events (receiving a request or a response).
- |~| : if the choice between these transitions is an internal choice. In other words, if these Pci are initialized events (sending a request or a response).

B. Assembly connectors translation

Similarly to the Wright connector, an UML2.0 assembly connector represents a communication link between two elements (e.g., ports or interfaces). Hence, we propose to translate an UML2.0 assembly connector to a Wright connector. The name of the connector is obtained by the concatenation of the names of the two interconnected components. This connector must have two roles:

- The name of the first role is obtained by the concatenation of the " R " character (to say Role) with the port name of the first component.
- The name of the second role is obtained by the concatenation of the " R " character (to say Role) with the port name of the second component.

To ensure the validation of the fourth property related to the consistency of the attachment Port/Role (see Section 3), we propose to create the CSP process modeling the behavior of each role by the same CSP process specifying the behavior of the port attached to this role. The CSP process modeling the Glue (the overall behavior of the connector) will be written by a process compatible with its roles. This ensures the second property (connector without deadlock). For this, we propose to associate for each called method in the role, the following process:

```
Receiving a request -> sending this request ->
receiving a response to this request -> send-
ing of this response.
```

Example:

```
Connector BankAtm
Role Client=_verifyPin->RverifyPin-> TICK
Role Server=verifyPin->_RverifyPin-> TICK
Glue= Client.verifyPin->_Server.verifyPin  ->
Server.RverifyPin->_Client.RverifyPin->TICK
```

C. Component instance diagramme translation

An instance diagram of UML2.0 components is translated to a Wright configuration according to the following correspondence:

- Any UML2.0 component instance is translated to an adequate Wright component instance;
- Any assembly of two UML2.0 component instances is translate by:
  - -An instance of the assembly connector that relates these two components;
  - -An attachment of the port of the first component with the first role of this connector and,
  - -An attachment of the port of the second component with the second role of this connector

V. Validation on ATM/Bank system

The ATM/Bank is a complex system. This study does not seek to explain its detailed operation, but to illustrate some of the main concepts of a component-based system. Our system consists essentially of two components: an ATM that represents an automated silver teller machine, and a Bank component representing a subsystem of a bank. The operating system of these components is as follows:

When inserting a bank card into an ATM, it must authenticate itself by sending a PIN code and the IBAN code of the card to the appropriate bank. At this time, the bank checks the validity of this information. If the codes are correct, the Bank component allows the ATM component to consult and/or debit from this account. To simplify this system, we ignore the part of ATM communication with the user.

A. Modeling In UML2.0/PoSM

In this section, an UML2.0 specification of the ATM/Bank system is presented. Figure 2 shows the UML2.0 components diagram modeling the static view of the system.

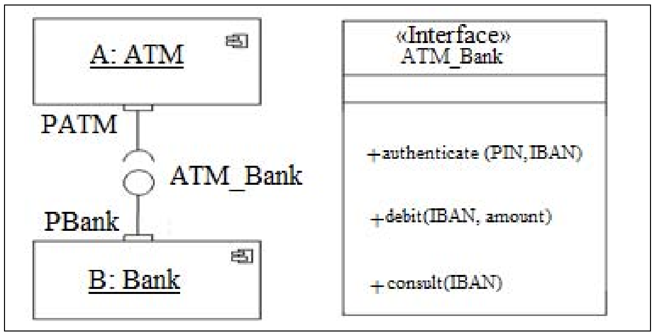


Figure 2. Component diagram of ATM/Bank system

The ports behavior and the overall behavior of each component of this system are specified in Figure 3 by state machines using the PoSM profile.

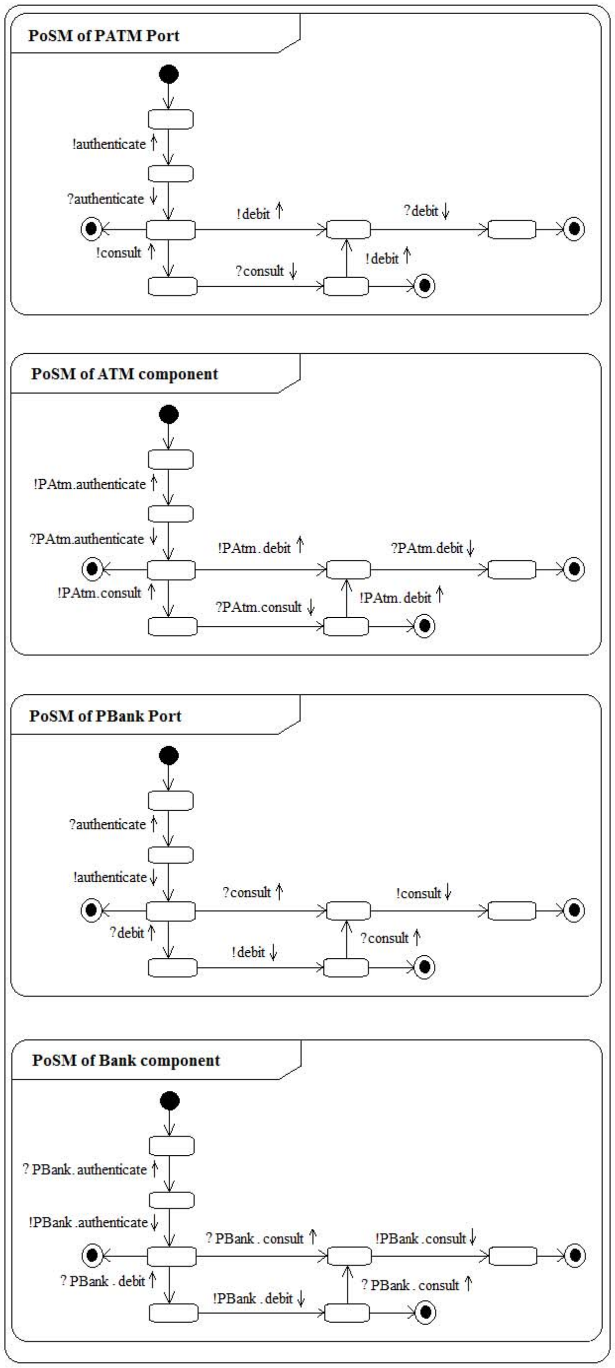


Figure 3. PoSMs of ATM/Bank components

B. *Translation into Wright/CSP*

Figure 4 shows a Wright configuration obtained by using our translation rules on the UML2.0/PoSM software architecture. Using the Wr2fdr tool on the target Wright configuration, we got a CSP description equipped with a set of contracts. These contracts aimed the behavioral port/role compatibility and compatibility between the component's ports and the component itself.

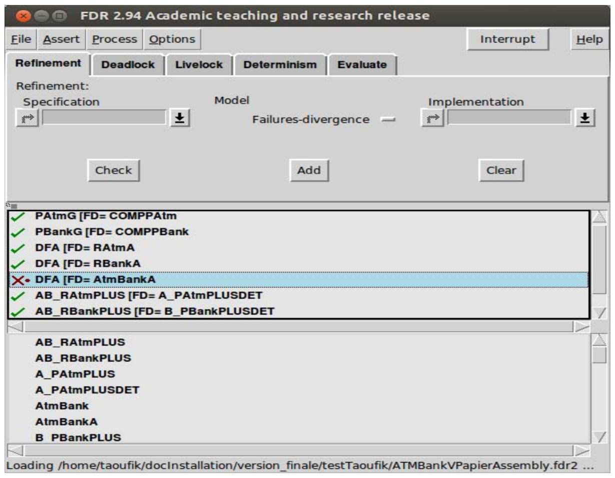
```
Configuration ATMBANK

Component Atm
  Port PAtm = _authenticate -> Rauthenticate -
> (TICK |~| _debit -> Rdebit -> TICK |~|
_consult -> Rconsult -> (TICK |~|_debit ->
Rdebit -> TICK ))

  Computation = _PAtm.authenticate ->
PAtm.Rauthenticate -> ( TICK |~|PAtm.debit ->
_PAtm.Rdebit -> TICK |~| _PAtm.consult ->
PAtm.Rconsult -> (TICK |~| PAtm.debit ->
_PAtm.Rdebit -> TICK))

Component Bank
  Port PBank =authenticate -> _Rauthenticate -
> (
TICK [] consult -> _Rconsult -> TICK []
debit -> _Rdebit -> (TICK [] consult ->
_Rconsult -> TICK ))

   Computation = PBank.authenticate ->
_PBank.Rauthenticate -> (TICK [] PBank.consult
-> _PBank.Rconsult -> TICK [] PBank.debit ->
_PBank.Rdebit -> (TICK [] PBank.consult ->
_PBank.Rconsult -> TICK ))

   Connector AtmBank
Role RAtm = _authenticate -> Rauthenticate ->
(TICK |~| _debit -> Rdebit -> TICK |~|
_consult -> Rconsult -> (TICK |~|_debit ->
Rdebit -> TICK ))

Role RBank = authenticate -> _Rauthenticate ->
(
TICK [] consult -> _Rconsult -> TICK []
debit -> _Rdebit -> (TICK [] consult ->
_Rconsult -> TICK ))

 Glue = RAtm.authenticate -> _RBank. authenti-
cate -> RBank.Rauthenticate ->
_RAtm.Rauthenticate -> (TICK [] r1 [] r2)
       where {
         r1=RAtm.consult-> _RBank.consult ->
RBank.Rconsult -> _RAtm.Rconsult -> (TICK []
r2)
         r2=RAtm.debit-> _RBank.debit ->
RBank.Rdebit -> _RAtm.Rdebit -> (TICK [] r1)
       }

  Instances
       A  : Atm
       B  : Bank
       AB : AtmBank

   Attachments
       A.PAtm As AB.RAtm
       B.PBank As AB.RBank

End Configuration
```

Figure 4. Wright specification of ATM/Bank system

C. Cheking with FDR2

Figure 5 shows that the consecutive use of the Wr2fdr and FDR2 tools on the target Wright configuration allows concluding that although the source UML2.0/PoSM software architecture appears correct from standpoint syntactic consistency between required and offered interfaces, it is nonetheless incorrect from a behavioral standpoint. This occurs as the PAtm port of the ATM component is incompatible with that of the Bank component. Indeed, although both ports seem similar, the first accepts the sequence of execution methods (consult after debit), while the second does not allow it.

Two other versions of the behavior Bank component that are compatible with that of the ATM component are available in the SourceForge web site.[1]



```
Refinement:
Specification            Model                      Implementation
                         Failures-divergence

✓ PAtmG [FD= COMPPAtm
✓ PBankG [FD= COMPPBank
✓ DFA [FD= RAtmA
✓ DFA [FD= RBankA
✗ DFA [FD= AtmBankA
✓ AB_RAtmPLUS [FD= A_PAtmPLUSDET
✓ AB_RBankPLUS [FD= B_PBankPLUSDET

AB_RAtmPLUS
AB_RBankPLUS
A_PAtmPLUS
A_PAtmPLUSDET
AtmBank
AtmBankA
B PBankPLUS

Loading /home/taoufik/docInstallation/version_finale/testTaoufik/ATMBankVPapierAssembly.fdr2 ...
```

Figure 5. Checking the CSP description of ATM/Bank system

VI. Conclusion

In this paper, a contribution related to the detection of the non-consistency of UML2.0/PoSM software architecture behavioral properties has been proposed. Our contribution covers the behavioral properties related to component and connector consistency, and port/role compatibility proposed by Wright. The verification of these properties is entrusted to FDR2 model-checker through the Wr2fdr translator able to translate a Wright configuration to a CSP specification.

In our future works, we will automate the UML2.0/PoSM to Wright/CSP transformation using an MDE approach. In addition, the Wright/CSP target specification can be transformed into an Ada concurrent program to check the specific behavioural properties of UML2.0/PoSM software architecture. In fact, the semantic rapprochement between a CSP process and an Ada task favours the realization of a Wright/CSP abstract architecture by an Ada concurrent program. Using an Ada formal analyses tool such as FLAVERS, we can detect the potential deadlocks of an Ada program.

[1] https://sourceforge.net/projects/posm-of-atm-bank-system/

References

[1] J. Jacobs and A. Simpson, On a Process Algebraic Representation of Sequence Diagrams, SEFM Workshops 2014, Grenoble, France.

[2] T. Lambolais, A. L. Courbis, H. V. Luong. Raffinement de modèles comportementaux UML, vérification des relations d'implantation et d'extension sur les machines d'états. 24 èmes Journées francophones d'Ingénierie des Connaissances, Jul 2013, Lille, France.

[3] F. Jouault, F. Allilaire, J. Bézivin , I. Kurtev. Atl: A model transformation tool. Sci. Comput. Program. 2008, vol. 72, pp. 31-39.

[4] V. Mencl. Enhancing Component Behavior Specifications with Port State Machines. Tech. Report No. 2003/4, Dept. of SW Engineering, Charles University, Prague, September 2003.

[5] Ltd, Formal System (Europe) Failure-Divergence Refinement, FDR2 user Manual, May 2003.

[6] F. Oquendo. Pi-adl : an architecture description language based on the higher-order typed pi-calculus for specifying dynamic and mobile software architectures, ACM Software Engineering Notes 2004, vol. 29, pp. 1-14.

[7] ArchWare Consortium, 2001. The EU funded ArchWare – Architecting Evolvable Software - project : http://www.arch-ware.org

[8] D. Sangiorgi. Expressing Mobility in Process Algebras : First-Order and Higher-Order Paradigms. PhD thesis, University Edinburgh, UK, February 1992.

[9] Alloui I., Garavel H., Mateescu R., Oquendo F., The ArchWare Architecture Analysis Language. ArchWare Project IST-2001-32360 Deliverable D3.1b, 2003.

[10] E. Leroux, F. Oquendo, Q. Xiong. Test de conformité basée sur l'architecture logicielle. CAL 2013, May 2013, Toulouse.

[11] D. Giannakopoulou. Model Checking for Concurrent Software Architectures. PhD thesis, Imperial College of Science, janvier 1999.

[12] J. Magee and J. Kramer. Concurrency. State Models and Java Programs. wiley, 1999.

[13] Chang H., Collet P., « Eléments d'architecture pour la négociation de contrats extatsfonctionnels». 1ère Conférence francophone sur les Architectures Logicielles (CAL 2006), Nantes, France Hermès Science. Septembre 2006, pp. 151-167.

[14] P. Collet, R. Rousseau, T. Coupaye, and N. Rivierre. A contracting system for hierarchical components. In George T. Heineman, Ivica Crnkovic, HeinzW. Schmidt, Judith A. Stafford, Clemens Szyperski, and Kurt C. Wallnau, editors, CBSE, volume 3489 of Lecture Notes in Computer Science, pages 187–202. Springer, 2005. ISBN : 3-540-25877-9.

[15] T. Barros, L. Henrio, and E. Madelaine. Behavioural models for hierarchical components. In Proceedings of SPIN'05. Spinger Verlag, 2005. To appear.

[16] As-2 Embedded Computing Systems Committee SAE. Architecture Analysis & Design Language (AADL). SAE Standards no AS5506, novembre 2004.

[17] Language Compliance and Application Program Interface. SAE, juillet 2006. http://www. sae.org.

[18] T. Vergnaud, L. Pautet, F. Kordon, Using the AADL to Describe Distributed Applications from Middleware to software Components. Ada-Europe 2005, Springer-Verlag, Berlin, 2005.

[19] M. Y. Chkouri. Modélisation des systèmes temps réel embarqués en utilisant AADL pour la génération automatique d'applications formellement vérifiées. Thèse de doctorat, Université Joseph Fourier, 07 Avril 2010

[20] F. Plasil and S. Visnovsky, Behavior Protocols for Software Components. Transactions on Software Engineering, IEEE, vol 28, no 11, November 2002

[21] R. Allen, A Formal Approach to Software Architecture. Phd Thesis, Carnegie Mellon University, School of Computer Science. May 1997.

[22] A. Garis, C. R. Paiva, A. Cunha and D. Riesco, Specifying UML Protocol State Machines in Alloy. 9th International Conference, IFM 2012, Pisa, Italy, 2012.

[23] C. A. R. Hoare, Communicating Sequential Processes, Prentice Hall International, 2004.

[24] M. Y. Ng and M. Butler, Towards Formalizing UML State Diagrams in CSP. 1st International Conference on Software Engineering and Formal Methods (SEFM 2003), Brisbane, Australia. September 2003.