Validation By Animation : Animating Software Architectures Based On the π-calculus

Selma AZAIEZ*, Frédéric POURRAZ*, Hervé VERJUS*, Flavio OQUENDO**

*LISTIC-ESIA Lab.
University of Savoie
B.P. 806 - 74016 Annecy Cedex – France
E-mails : {selma.azzaiez, frederic.pourraz}@etu.univ-savoie.fr
herve.verjus@esia.univ-savoie.fr

** ESIA - University of Savoie
B.P. 806 - 74016 Annecy Cedex – France
E-mail : flavio.oquendo@univ-savoie.fr

Abstract:
This paper introduces an animation approach for validating software architectures. This approach is dedicated to architectural descriptions expressed using π-calculus-based ADLs. We based our approach on semantics analysis in order to conserve correctness between the architectural description and the animation rendering. The purpose of our work is to focus on the capability of animation to provide a simplified view that facilitates the architectural validation and also the understanding of the ADL notation.

I. Introduction

This paper is focusing on the software animation that is one of software visualisation approach. We employ this technique in the context of architecture centric development process which places the software architecture at the heart of the development process. Architecture Description Languages (ADLs) have been proposed as modeling notations [6] to describe software architectures. Many tools can be provided with ADLs, the most current among them are analysis tools, model checkers, parsers, compilers, runtime support tools, and so on. Some of ADLs provide formal syntaxes and semantics that can be based on algebraic process such as π-calculus.
Although animation was widely exploited in program animation [1][2][3] and even in program specification [4], few works concerning ADL environments offer support to animate the described architectures.

However, we think that animation can be useful to assist the user (the software architecture designer) in the analysis, design and understanding of software applications. We are especially interested on the formal ADL notation since that their use requires some expertise. The purpose of our work is to focus on the capability of animation to provide a simplified view that facilitates the use of these ADL notations. π-calculus [5] will be used in our experimental environment. After depicting the different animation approaches applied until now (section III), we will explain our approach and the underlying mechanisms in animating software architecture (section IV). These mechanisms aim at capturing the semantics of π-calculus code while executing it. Then, we will show how we plug-in animation rules to this mechanism. We finally briefly describe the prototype implementing our approach.

II. Architecture-centric development process

The software system architecture reflects some of the most basic and most important design decisions. In architecture-centric development processes (figure 1), the architecture is treated as a central product. It identifies the problem domain abstractions, their interactions and dependencies. Software architecture has considerable impact on the progress of the software project since architectural decisions are among the first ones to be taken during system development, and since they affect all later stages of the development process. The architectural mistakes often have economical consequences. They are as important as architectural mistakes can lead to change again all the development process. The architecture-centric development process aims at detecting architectural problems (indeterminism, misunderstanding, non-conformance, consistency and properties violation, etc.) at a very early stage of the development process. To cover the architecture-centric process (or a part of it), *Architecture Description Languages* (ADLs) and accompanying tools have been proposed [6]. ADLs have recently become an area of intense research in the software architecture community. There is, however, little consensus in the research community on what an ADL is and what architectural aspects should be modeled and covered. Some ADLs only support the structural aspects of the architecture while others support both the structural and behavioral aspects.

An ADL environment provides some tools supporting analysis, refinement, code generation and dynamism etc. These tools are provided according to the ADL area of interest. For example, Wright [9] which focuses on analysis, provides a parser and a model checker for analysing type conformance. Animation can support some categories as shown in the following section.

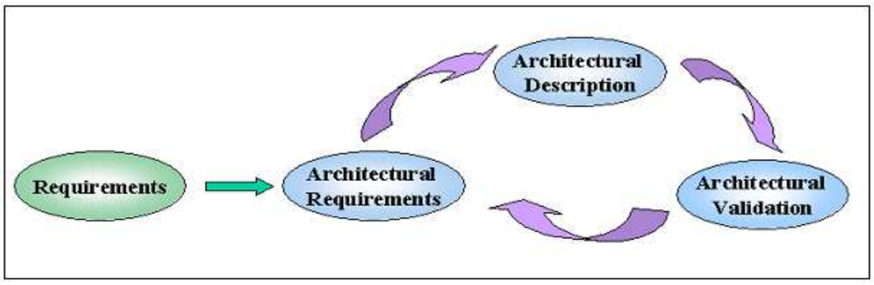


Figure 1. Architecture-centric development process

III. Using Animation in architecture-centric development process

Among tools that can support ADLs, there are animators which can be used for different purposes. The most appropriate one is to provide multiple views of the executing architecture descriptions. However, animation can also be used in active specification, analysis, or even architectural evolution.

In [7], authors introduce an approach consisting in associating a behavioral description with a structural one expressed using Darwin, in order to analyse the overall system architecture. The behavioral description is provided by the Labeled Transition Systems. These diagrams may be animated, according to interactive simulations in order to check whether or not the specification corresponds to the intuition of the architect. Other ADLs provide animation tools for analysis purpose. Rapide [8] for example provides a simulation tool and an animator showing the architectural execution.

Other ADLs provide animation tools for evolution purpose. C2 [9] provides *Extension Wizard* that simulates the end-user interface which depicts the architecture execution. This interface allows also the modification of the architecture at runtime. This is done by dynamically loading and linking new components or connectors into the architecture description. Weaves [10] adopts the same approach: it allows insertion of low-overhead observers into a weave for supporting real-time animation execution. The weaves animation tool allows also unanticipated dynamic manipulation of the described architecture. Thus, many issues are tackled in software architecture animation. The purposes of animation are numerous. Although the means for dynamic view of the architecture, animation can be used for analysing, active specification and architectural evolution.

In this paper, we propose an animation approach that can be used for analysing and validating architecture. This work is realized within the

ArchWare European project[1]. This project aims at providing a software architecture centric engineering environment, supporting dynamic evolution of software systems architectures. It allows to formally specify the architectures of software systems, to refine architectures into compliant implementations. It also allows to deploy, operate and maintain these architectures. ArchWare environment provides a formal language called π-ADL [11] and a set of associated tools. π-ADL is based on π-calculus [5]. Our approach consists in providing an animation tool that can support the analysis of formal languages based on π-calculus. Although the numerous advantages offered by this kind of formal languages [12] (e.g. precision, unumbiguous etc.), they are not much employed. This is due to widespread perception among software developers that formal notations and formal analysis techniques are difficult to understand and apply. Thus, we propose to provide an animation tool to assist the architect in understanding the formal construct semantics. In the following, we present our approach and the implemented prototype.

IV. Animation approach for π-calculus based ADLs

The animation tool must graphically interprete the π-calculus language. It provides a structural and behavioral view of the specified architecture. This view is useful helping the architect on validating its architecture in an intuitive manner. To extract this view, we first need a mechanism to interpret and execute the ADL. Then, ADL semantics is depicted as graphical animation. Thus, the animation tool has two principal functionalities. The first one is the execution and the interpretation of the described architecture. The second functionality concerns the graphical displaying and animation. For the semantic interpretation, we use MMC (Mobile Model Checking) [13] implemented using Prolog language. Indeed, we use Java for graphical displaying.

IV.1. MMC

MMC is a mechanism that offers an operational semantics to analyse and execute π-calculus descriptions. The salient issues that arise in π-calculus include scope extrusion and intrusion, and dynamic generation of new names to avoid name capture. To address these issues and implementing them, MMC uses the logic programming. This one allows to

[1] IST European Project ArchWare Architecting Evolvable Software (Project N°IST-2001-32360) – www.arch-ware.org

obtain an exact encoding of the π-calculus transitional semantics. The MMC mechanism is separated into three parts:

- the translation of π-calculus language syntax towards MMC syntax,
- the operational semantics allowing to analyse and execute π-calculus descriptions,
- the formal properties verification.

MMC offers a syntax allowing to express π-calculus processes. This syntax is a logical formalism expressed as Prolog facts. Operational semantics are used to capture MMC description semantics while execution. These two functionalities are useful in our animation approach. Indeed, the formal properties verification is useless for the animation process since they propose a formal complicated analysis such as deadlock checking, data lost checking etc. This one contrasts with the intuitive views that animation should provide. Actually, the qualities attributes reasoning should be considered as a complementary of the animation. As these rules are implemented regardless of operational semantics (in a separated files), it was easy to omit them. Instead, we plug-in animation rules that will carry out the animation analysis by identifying the graphical sequences that should be displayed.

IV.1.1 Towards MMC syntax

MMC begins by translating π-calculus process expressions in MMC syntax. Let us take a π-calculus expressions subset and show their translation in MMC. This subset is given in figure 2.

$$\begin{array}{lcl} \alpha & ::= & x(y) \;\mid\; \bar{x}y \;\mid\; \tau \\ P & ::= & 0 \;\mid \alpha.P \mid P \mid P \end{array}$$

Figure 2. π-calculus expressions subset

α denotes a set of prefixes and P a set of process expressions. Prefixes represent respectively input, output and internal actions. P is composed by 0 which is the process with no transitions, α.P which perform an α action and then behaves as P and then P|P which represents a parallel composition. MMC uses a one-to-one function that maps standard π-calculus syntax to process expressions and actions in MMC's syntax. This function is defined in figure 3.

$$\begin{aligned}
f_\theta(\mathtt{zero}) &= 0\\
f_\theta(\mathtt{tau}) &= \tau\\
f_\theta(\mathtt{in}(X_1, X_2)) &= \theta(X_1)(\theta(X_2))\\
f_\theta(\mathtt{out}(X_1, X_2)) &= \overline{\theta(X_1)}\theta(X_2)\\
f_\theta(\mathtt{par}(P, Q)) &= f_\theta(P) \mid f_\theta(Q)
\end{aligned}$$

Figure 3. MMC Syntax

We experiment our approach by applying it on the π-ADL [11] core (π-ADL is a layered language) that is based on π-calculus. According to the functions previously described, we developed a parser allowing the translation between π-ADL core and MMC syntax. The result of the parsing (which is expressed in MMC syntax) is given as Prolog facts *"def(Z,K)"* where Z is the process identifier and K the process actions. Here after we give a result of the traduction of π-ADL core expressions in the MMC expressions. We present the exemple of a client process which try to connect a server.

```
value client is abstraction ().
{
 value call, wait is
       free connection[].
 via call send.
 via wait receive.
 unobservable.
 }.
```

```
def ( client,


  pref ( out ( "call", "null"),
       pref ( in ( "wait", "null"),
            pref ( tau, zero)))).
```

Figure 4. From π-ADL expressions to MMC expressions

IV.1.2 MMC semantics analysis

MMC process expressions are analysed using the Prolog predicates named "*trans*" relation. The "*trans*" relations are a direct encoding of the symbolic semantics of π-calculus. This is very useful for our animation process since it offers an automatic mechanism to capture the π-calculus semantics of the described architecture.

Symbolic semantics of π-calculus are usually denoted by $P \xrightarrow{M,\alpha} P'$ where P and P' are π-calculus process expressions, α is an action and M is the constraints that describe the equality between names under which the transition is enabled. The *"trans"* relations that encode this semantics are of the form of *trans(P1 ,A, M, P2, Nin, Nout).* It means that process expression *P1* can evolve into process expression *P2* via the execution of an *A* action provided that the set *M* of equality constraints over names hold. *Nin* and *Nout* are integers used to generate new names. Representing π-calculus names as Prolog variables in MMC enables to treat scope intrusion and extrusion, renaming and namer restriction. Let's take the sending and receiving actions as example. These actions are analysed using these *"trans"* relation.

```
% SEND-ACTION:
trans(pref(send(X, Y), P), send(X1, Y), V, N, P) :-
((nonvar(X), nonvar(X1)) -> X==X1, V = [];
                              X=X1, (N= =1 -> V = [X];
                                                         V = [])).
% RECEIVE-ACTION:
 trans(pref(receive(X, Y), P), receive(X1, Y), V, N, P) :-
 ((nonvar(X), nonvar(X1)) -> X==X1, V = [];
                               X=X1,(N==1 -> V = [X];
                                                 V = [])).
```

According to the previous code, we make tests on X and X1 (which represent connection values). *((nonvar(X), nonvar(X1))* test is true if X and X1 are instantiated. In this case, if they are equals, *send(X1,Y)/receive(X1,Y)* actions are performed and the communication is synchronous. If the precedent tests fail, X1 value is allocated to X. Then, another test is performed on N value. If N is equal to 1, the communication is asynchronous and X is affected to V list while waiting the corresponding send/receive action. This example shows the importance of the semantics analysis before the animation visualisation. This analysis identifies if the sending/receiving action is synchronous or asynchronous. Accordingly, the animation rendering will be different. Aside from encoding the symbolic semantics of π-calculus, the *trans* relation permits also the π-calculus description execution. The predicate that activates the transition to another action is expressed here after:

```
transition(P1, A, P2) :- trans(P1 , A ,_ , 0, P2).
```

For our animation process, these transitions ensure the continuity of the animation. Thus, starting from MMC mechanism leads to a coherent and correct animation rendering.

IV.2. Animation rules

A mechanism defines a set of rules that identifies the graphical animation to be displayed for a given expression. The goal of these rules is to establish the link between an architectural description and its corresponding animation. It includes the refinement stages of the initial architectural description towards the generation of the final animation.

MMC is used for the identification and analysis of the expressions that have to be animated. An animation analysis is necessary to identify the animation sequences that must be displayed according to each expression. For this purpose, we plug-in an animation rule to each "*trans*" relation. The animation rule fires animation pattern. These patterns identify the graphical elements and animation sequences that can be carried out on these elements. Thus, animation rules act as an interface between the semantic interpretation of the ADL and the animation visualisation. As semantic interpretation is encoding using Prolog,

animation visualisation must be implemented using a language supporting animation rendering. In our case, we choose Java.
Animation rules have exactly the same form as the *trans* relation which facilitate their plug-in. They are activated at the end of the *trans* relation. This rules fired the corresponding animation patterns that graphically execute the corresponding semantic action. At the end of the animation rule, *trans* relation is called which permits to continue the execution. For example, the *trans* relation and animation rule corresponding to the choice action are expressed in the following :

```
% trans relation:
trans(choice(P, Q), A, V, N, P1) :- animationRule(P, A, V, N, P1).
trans(choice(P, Q), A, V, N, Q1) :- animationRule(Q, A, V, N, Q1).

% Animation rule
animationRule(choice(P, Q), A, V, N, PQ) :-
        animator_object(Object),
        javaMessage(Object, animator_object(Object)),
        javaMessage(Object, choosePattern(string(C))),
        if (C ==’’0’’) ->  trans(choice(P, Q), A, V, N, P);
                                 trans(choice(P, Q), A, V, N, Q).
```

One can note that animation rule connect the *trans* relation to animation pattern that describes the corresponding graphical animations. We use Java for implementing graphical aspects. The Prolog facts *animator_object (Object)* and *javaMessage (Object, animator_object(Object))* refer to the animator java object which implements the *animator-object* in the java side. The communication with java is possible using InterProlog [14]. InterProlog is a library allowing the development of combined Java and Prolog applications. A *javaMessage(Target,Message)* predicate provided by InterProlog allows Prolog to call any java method. *JavaMessage (Object, choosePattern(string(C)))* permits to call the *choosePattern* method that is encoded in java. The activated pattern will interpret the semantic of the choice by displaying a GUI that permits the user to choose to continue with *P* or *Q*.
Moreover, the animation rules complete the MMC analyse. Let us take the example of sending and receiving messages. In this case, animation analyse can be done by different steps. For example, for a synchronised communication between parallel processes, we start by analysing independently the send and receive actions. This is done respectively to the corresponding *trans* relation. After this preliminary analysis, the forward action is then deducted and “*forwardRule*” is called. The “*forwardRule*” activates the pattern corresponding to the message forward.

IV.3. Animation patterns

As previously discussed, we depict the graphical visualisation in the animation patterns. They contain the graphical elements (representing the structure) and the animation actions (representing the behaviour) that can be arise on these elements. The animation rules fire the patterns. Each pattern encodes the graphical visualisations corresponding to each expression identified by the animation. Moreover, as animation analyse can be done by different steps, graphical visualisation is also constructed in different steps and thus depicted in several patterns levels. Let us return to the example of the synchronised communication (see illustration in the figure 5):

- The first "low level" pattern associated to the fact that a message is on hold of being sent is activated (figure 5.a). It identifies the animation element (a round representing the message) and the animation action (i.e. flickering of the message to be sent),
- on the same way, the second "low level" pattern (receiving pattern) that is associated to the fact that a message is on hold of being received is activated (figure 5.b). For example, the animation element is a line representing the connection and the animation action is flickering of the connection on which the message must be received;
- once these two patterns are activated, the sending and the receiving actions can be carried out. The "higher level" pattern (called the *forward* pattern) is activated. It supports the message to move on the connection between the client and the server (figure 5.c).

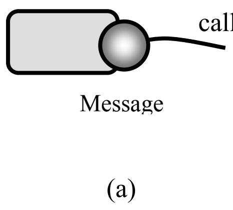
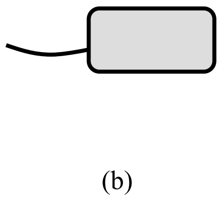
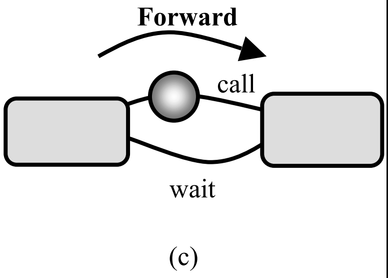
MMC processes	Sending pattern	Receiving pattern	Forward Pattern
def (client, **pref (out (** "call", "signal"**),** **pref (in (** "wait", "signal"**),** **pref (tau, zero)))).** **def (**server, **pref (in (** "call", "signal"**),** **pref (tau,** **pref(out (** "wait", "signal"**),** **zero)))).**			

Figure 5. Animation patterns rendering

V. Concluding remarks

This paper introduces a tool for animating software architectures, described by using π-calculus-based ADLs. One of the issue raised in the animation domain is the language semantic interpretation and the mapping with a corresponding graphical visualisation. We propose an approach that satisfies these properties. We claim that our approach is generic enough to be applied with π-calculus based ADLs. To preserve the semantic while running a π-calculus description, we use the *trans* relations provided by the MMC system. We plug-in animation rules that preseve the semantic while displaying the graphical visualisation. The graphical visualisation is depicted in animation patterns. Animation patterns can be considered as a generic graphical visualisation technique that can be used for any ADL. They are fired at the end of the animation process when the semantic analysis has been done. A prototype has been constructed whithin the ArchWare project. The development of this prototype is realized using different tools. XSB Prolog is used for the semantics analysis and the execution of the architecture. InterProlog [14] and Java are used for implementing the graphical rendering.

The MMC mechanism handles the moniadic π-calculus semantics. It was also used to check the semantics of poliadic π-calculus and μ-calculus [5]. Thus, it is possible to use it in order to address other languages based on this notation and that have a more complex semantics. The prototype actual release covers the π-ADL core. It is experimented in the ArchWare project, especially for animating an industrial EAI solution architecture, involving legacy systems and COTS (see [15]).

References

[1] Stasko J.T., *“Tango: A Framework and System for Algorithm Animation”*, IEEE Computer, Vol. 23(9), pp. 27-39, 1990

[2] Marc H. Brown and Robert Sedgewick. *‘‘A system for algorithm animation’’*, Computer Graphics, 18(3):177–186, 1984.

[3] Heath M., Finger J.E., *“ParaGraph: A Tool for Visualizing Performance of Parallel Programs”,* The National Center of Supercomputing Applications, University of Illinois at Urbana-Champaign.

[4] G. Reeve, S. Reeves, *“Experiences using Z animation tools”*, Working Paper Series ISSN 1170-487X.

[5] R. Milner J. Parrow, D. Walker, *“A Calculus of Mobile Processes Pt.1”*, Information and Computation 100(1) pp.1-40- September 1992

[6] Medvidovic N., Taylor R. “*A Classification and Comparison Framework for Architecture Description Languages*”, Technical Report UCI-ICS-97-02, Department of Information and Computer Science, University of California. Irvine, February 1997.

[7] J. Magee, J. Kramer, D. Giannakopoulou, *“Behaviour Analysis of Software Architectures”,* Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1), pages 35-50, 1999.

[8] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. “*Specification and Analysis of System Architecture Using Rapide.*” IEEE Transactions on Software Engineering, pages 336-355, April 1995.

[9] P. Oreizy, N. Medvidovic, R. N. Taylor, *“Architecture-Based Runtime Software Evolution”,* in the Proceedings of the International Conference on Software Engineering 1998(ICSE’98). Kyoto, Japan, April 19-25, 1998. http://www.ics.uci.edu/~peymano/

[10] M. M. Gorlick and R. R. Razouk. *“Using Weaves for Software Construction and Analysis”,*In Proceedings of the 13th International Conference on Software Engineering(ICSE13), pages 23-34, Austin, TX, May 1991.

[11] F. Oquendo, I. Alloui, S. Cimpan, H. Verjus, “*Final Definition of Abstract Archware/Core ADL and Style ADL”,* Deliverable D1.1b, ArchWare project, March 2003.

[12] Heimdahl, M.P.E., Heitmeyer, “C.L.: *Formal Methods for Developing High Assurance Computer systems: Working Group Report”,* Proceedings, Second IEEE Workshop on Industrial-Strength Formal Techniques (WIFT'98), Boca Raton, FL, Oct. 19, 1998.

[13] P. Yang, C.R. Ramakrishman, Scott A. Smolka, “*A Logical Encoding of the π-calculus: Model Checking Mobile Processes Using Tabled Resolution*”, Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation, Jan. 2003, New York.

[14] M. Calejo, “*InterProlog: a declarative Java-Prolog interface*”, in Proceedings, Logic Programming for Artificial Intelligence and Information Systems (thematic Workshop of the 10th Portuguese Conference on Artificial Intelligence), Porto, 2001.

[15] L. Blanc dit Jolicoeur, C. Braesch, R. Dindeleux, S. Gaspard, D. Le Berre, F. Leymonerie, A. Montaud, C. Chaudet, A. Haurat et F. Théroude, “*Final Specification of Business Case 1, Scenario and Initial Requirements*”, Deliverable D7.1b, ArchWare project, December 2002.