A relationship between sequence and statechart diagrams.

Yves Dumond (*), Didier Girardet (*), Flavio Oquendo (**)

LLP/CESALP Laboratory – University of Savoy
(*) Campus Scientifique F-73376 Le Bourget-du-Lac Cedex
(**) ESIA B.P. 806 F-74016 Annecy Cedex
Yves.Dumond@univ-savoie.fr, Didier.Girardet@edf.fr, Flavio.Oquendo@univ-savoie.fr

Abstract. This paper is an introduction to an easy way to formalise sequence diagrams and verify coherence with the statechart diagrams, with the pi-calculus that is a process algebra where processes interact by sending communication links to each other.

Research motivation

It is currently recognized that the Unified Modeling Language (UML) [1] suffers from its lack of formal semantics [2, 3]. The current semantics of UML is made up of a metamodel that describes the notation and informal textual annotations. The Object Constraint Language (OCL) completes the syntactic rules of the metamodel and improves the precision of the notation. But it is not enough to check and to validate UML models. One way to make the UML semantics more precise is to combine semi-formal and formal notations [4, 5]. The integration of the two approaches is synergetic. Many works relate to static behaviour of UML models. But not many works relate to the dynamic behaviour. This paper describes a way, based on mathematical paradigms, to formalise sequence diagrams and to check the consistency with statechart diagrams.

UML offers many different notations to represent dynamic behaviour, such as statechart diagrams, sequence diagrams, collaborations diagrams and activity diagrams, and they are based on different paradigms/techniques:

- *Sequence* diagrams show an interaction arranged in explicit time sequence of stimuli and are better for real-time specifications and for complex scenarios.
- *Collaboration* diagrams show an interaction organized around the objects playing different roles and their link to each other, and are better for understanding all of the effects on a given instance and for procedural design.
- *Statechart* diagrams describe possible sequences of states and actions through which the element can proceed during its lifetime as a result of reacting to discrete events. Used in situations where asynchronous events occur.
- *Activity* diagrams are a special case of a state diagram. Used in situations where all or lots of the events represent the completion of internally generated actions.

The relationship between these UML diagrams is not simple. But, it is important to precise the notation used in these diagrams and to verify the coherence of a dynamic model. A first step to verify the coherence between sequence diagrams and statechart diagrams consist to bind states in statechart diagrams with messages in sequence diagrams. In the sequence diagrams, objects interact with other objects by exchanging messages. And the stimuli trigger transitions, which take the receiver object from one state to another: We integrate the states of each object in the formal model of sequence diagram.

Objects, in sequence diagrams, interact to effect a desired operation or result. The general form of the collaboration is the concurrency. Over past years, many formalisms have been defined in order to model concurrency [6]. Among these, process calculi offer a smart algebraic and logical framework for the description and the validation of concurrent systems. This approach started with Robin Milner's Calculus of Communicating System (CCS) [7] and was in particular continued by its industrial counterpart, i.e. Lotos. More recently, the pi-calculus [8] has extended CCS in order to allow dynamic reconfiguration of systems. We argue that this algebra provides an appropriate context for modelling the sequence diagrams. Thus, we use the pi-calculus for the description of the dynamic behaviour of objects. Objects and states of objects are modelled as pi-calculus processes and messages along links are modelled as pi-calculus actions along pi-calculus communication channels.

An overview of the pi-calculus notation is necessary to understand the formalism in that sequence diagrams are translated. An example illustrates application of this method. Finally, some conclusions are discussed.

Introduction to the pi-calculus

The pi-calculus is a mathematical model of processes whose interconnections change as they interact. The basic computational step is the transfer of a communication link between two processes; the recipient can then use the link for further interaction with other parties. This makes the calculus suitable for modelling systems where the accessible resources vary over time. It also provides a significant expressive power since the notions of access and resource underlie much of the theory of concurrent computation, in the same way as the more abstract and mathematically tractable concept of a function underlies functional computation. This introduction of the pi-calculus is intended for a theoretically inclined reader who knows a little about the general principles of process algebra and who wishes to learn the fundamentals of the calculus and its most common and stable variants. We use the pi-calculus in its first order polyadic version [9], but with some minor syntactic modifications:

Let us first consider an example. Suppose a server, named *S*, and a client, named *C* whish use the server with a link, named *l*. In the pi-calculus this is expressed as follows: the server that sends data *x* along *l* is $\bar{l}$<x>; the client that receives some data *y* along *l* is l(y). After an interaction between the server and the client the system becomes:

$$\bar{l}\langle x\rangle.S\,|\,l(y).C \rightarrow S\,|\,C$$

Basic definitions

We assume a potential set of names N, ranged aver by a,b,…, z, which will function as all of the communication ports, variables and data values, and a set of processes ranged over by A. These processes can be in the following forms:

- The *nil process* does nothing, i.e. does not interact with the environment,
- An *Output Prefix* $\bar{l}$<x>.P. The intuition is that the name *x* is sent along the name *l* and thereafter the process continues as *P*. So $\bar{l}$ can be thought of as an output port and *x* as a datum sent from that port.
- An *Input Prefix* l(x).P, meaning that a name is received along a name *l*, and *x* is a placeholder for the received name. After the input the process will continue as *P* but with the newly received name replacing *x*. So *l* can be thought of as an input port and *x* as a variable, which will get its value from the input along *l.*
- A *Silent Prefix* ô.P, which represents a process that evolve to P without interaction with the environment.
- A *Sum* P+Q representing a process that can enact either *P* or *Q*. In particular, they cannot mutually interact. It is an indeterministic choice between the processes *P* and Q. The choice is made according to the context, i.e. the offers of interaction present in the environment when P+Q is executed
- A *Parallel Composition* P | Q, which represents the combined behaviour of *P* and *Q* executing in parallel. The components *P* and *Q* can act independently, and may also communicate if one performs an output and the other an input along the same port.
- A *Match* [x=y] P. As expected this process will behave as *P* if *x* and *y* are the same name, otherwise it does nothing.
- A *Mismatch* [x y] P. This process will behave as *P* if *x* and *y* are not the same name, otherwise it does nothing.
- A *Restriction* (õx).P. This process behaves as *P* but the name x is local, meaning it cannot immediately be used as a port for communication between *P* and its environment. However, it can be used for communication between components within P.

The pi-calculus is well defined to describe communications among objects.

Overview of formalisation with the pi-calculus

A sequence diagram shows a composition of objects exchanging messages, without describing the internal behaviour of these objects. We can schematise sequence diagrams as a set of processes exchanging messages. Concepts of pi-calculus are similar to this technique of collaboration. A pi-calculus process is defined with its interactions with the environment: actions passed on channels of communication. We consider an object will be formalised with a pi-calculus process. To preserve the object notation, we name these pi-calculus processes with the name of objects.

In a sequence diagram, a message is passed along the links between objects. In the pi-calculus, an action is passed along the channels between processes. We consider a message passed along a link will be formalised with information passed along a pi-calculus channel. To preserve the object view, the name of pi-calculus channels is the name of associations between classes in the class diagrams.

The statechart diagrams show a set of states that an object can take. The transition from one state to another state is triggered by an event, which can be a received message. And the description of this message is showed in sequence diagrams. The pi-calculus provides the possibility to associate a received message with a process. Therefore, we introduce the states, as a pi-calculus process, in the formal model of sequence diagrams. The method gives more abstraction to formal specifications, which facilitates their global comprehension and readability.

To illustrate our method, we propose a small well known example: We present a sequence diagram and a statechart diagram describing a simplified phone communication.

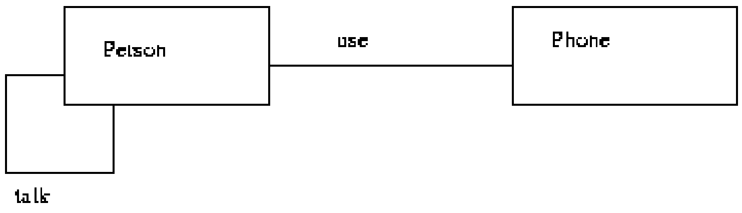

Fig. 1. Part of class *diagram, where the association between Phone and Person is represented.*

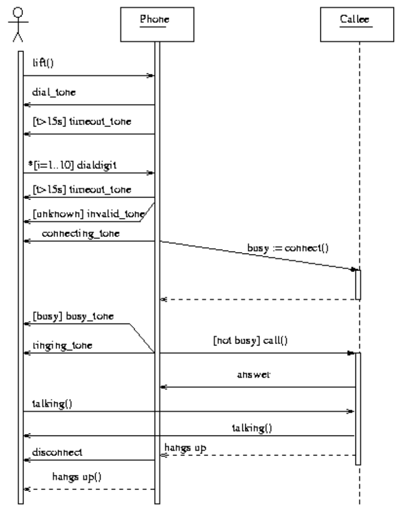

Fig. 2. A *sequence diagram representing a scenario of phone communication.*

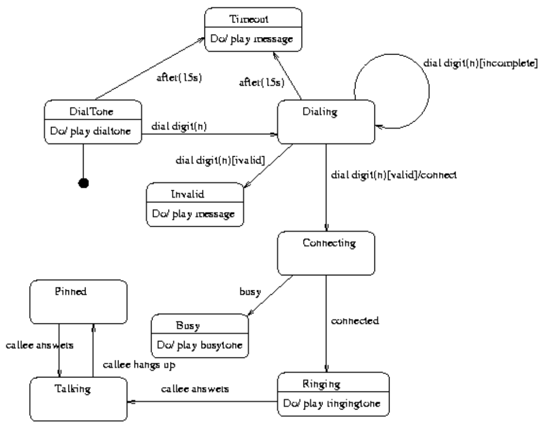

Fig. 3. S*tatechart diagram representing the different states that Phone can take.*

We apply our translation rules on the object *Phone* in the sequence diagram. These rules will be described in detail in the complete version of the paper. They are defined as Alexander pattern. Each concept used in sequence diagram is commented and a rule of translation is done with an example. So, we obtain the next formal specifications:

$$
\begin{aligned}
& Phone(prev\_state) \overset{def}{=} \\
& \quad [prev\_state = initial]\, use(lift).Dialtone \\
& + \\
& \quad [prev\_state = dialtone](d\ .Timeout + use(dialdigit).Dialing) \\
& + \\
& \quad [prev\_state = dialing]\,(e\ .Timeout + \\
& \qquad\qquad\qquad [valid = false]Invalid + \\
& \qquad\qquad\qquad [valid = true]\ \overline{use\_callee}\langle connect\rangle.Connecting) \\
& + \\
& \quad [prev\_state = connecting]\, use(busy).\ ([buzy = true]Busy + \\
& \qquad\qquad\qquad\qquad [busy = false]Ringing) \\
& + \\
& \quad [prev\_state = ringing]use\_callee(answer).Talking \\
& + \\
& \quad [prev\_state = talking]\, use\_callee(hangs\_up).Pinned \\
& + \\
& \quad [prev\_state = pinned]\, 0
\end{aligned}
$$

with

$$Dialtone \overset{def}{=} \overline{use}\langle dial\_tone\rangle.Phone(dialtone) \mid (\nu\ d\ )\overline{timer}\langle d\ ,15\rangle.0$$

$$Dialing \overset{def}{=} Phone\ (dialing\ ) \mid (\nu\ e\ \ )\overline{timer}\ \langle e\ ,15\rangle\ .0$$

$$Timeout \overset{def}{=} \overline{use}\langle message \rangle .0$$

$$Invalid \overset{def}{=} \overline{use}\langle message \rangle .0$$

$$Connecting \overset{def}{=} \overline{use}\langle connecting\_tone \rangle . Phone\,(connecting)$$

$$Busy \overset{def}{=} \overline{use}\langle busy\_tone \rangle .0$$

$$Ringing \overset{def}{=} \overline{use\_callee}\langle call \rangle \cdot \overline{use}\langle ringing\_tone \rangle . Phone\,(ringing)$$

$$Talking \overset{def}{=} Phone(talking)$$

$$Pinned \overset{def}{=} \overline{use}\langle disconnect \rangle \cdot \overline{use}\langle hangs\_up \rangle . Phone\,(pinned)$$

Conclusion

The primary objective of this research is to favour the integration of formal techniques in the actual practice of software engineering. This article has introduced a way to translate the UML sequence diagrams into the pi-calculus, by preserving the object paradigms. The formalisation is more intuitive and simple. The formal specifications are readable and comprehensible with abstraction of processes and without mathematical notation. All notations of the sequence diagrams can be taken into account with this approach. Moreover, a first formalisation of states is introduced and refinement is possible with other UML diagrams. The consistency between sequence diagrams and statechart diagrams is checked by verifying that the messages in the sequence diagrams trigger states in statechart diagrams.

This approach has deliberately taken the choice of the pi-calculus that is well adapted to formalise objects exchanging messages, described in sequence diagrams. A formal checking of properties of the specification can be carried out using the Mobility Workbench [10], but with some limitations. Moreover, the pi-calculus is supported by compilers as PICT [11] or TyCO [12].

This work demonstrates that it is possible to integrate semi-formal and formal methods for the dynamic behaviour of the UML models. This is a first step, and the next step in our further work should consist of completing the translation of statechart and activity diagrams.

Bibliography

[1] G. Booch, J. Rumbaugh, I. Jacobson: UML notation guide, version 1.1
Rational Software Corporation, Santa Clara, CA, 1 September 1997

[2] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe & V. Thurner
Towards a formalization of the Unified Modeling Language
Satoshi Matsuoka MehmetAksit, ECOOP'97 Proceedings, Springer Verlag, LNCS 1241, 1997

[3] A. Evans, R. France, K. Lano & B. Rumpe: Developping the UML as a formal modelling notation
UML'98, Mulhouse, 06/1998

[4] M. Gogolla & M. Richters: On combing semi-formal and formal object specification techniques
12th International Workshop, WADT'97, Tarquinia, Italy, 06/1997

[5] R. France, J.M. Bruel, M. Larrondo-Petrie & E. Grant: Integrating formal and informal notations
Proceedings of the 6th International AMAST Conference, 12/1997

[6] H. Saiedian: An invitation to Formal Methods
IEEE Computer, 24(4), 1996

[7] R. Milner: A Calculus of Communicating Systems
LNCS 92, Springer Verlag, 1980

[8] R. Milner, J. Parrow, D. Walker: *A Calculus of Mobile Processes*
Information and Computation, 100(1), 1-77, 1992

[9] R. Milner: *The polyadic pi-calculus : a tutorial*

W. Brauer and Schwichtenberg (editors) : Logic and Algebra of Specification, Springer-Verlag, 1993
[10] B. Victor: *The Mobily WorkBench Users'Guide, Polyadic version*
Internal report, Department of Computer Systems, Uppsala University, Swenden, 1995
[11] B.C. Pierce, D.N. Turner: Pict : A programming Language Based on the Pi-Calculus
Technical report IU, Indiana University, 1997
[12] V. Vasconcelos , M. Tokoro: *Core-TyCO, The Language Definition, Version 0.1*
Technical report DI/FCUL TR98-3, University of Lisbon, Portugal, 1998