Tutorial 2: A Foundation for Composing Concurrent Objects

Jean-Paul Bahsoun

IRIT - Universit Paul Sabatier
118 route de Narbonne
F-31062 Toulouse
email: bahsoun@irit.fr

In recent years, a lot of new languages and new concepts have been conceived in order to promote parallelism in the object-oriented framework. These proposals were investigated using different concepts related to parallelism and object orientation. Among these concepts, we can find shared variables/message passing, inheritance/delegation, reflection... The degrees of a good cohabitation may be appreciated by combining the above concepts. In order to have significant criteria we have to determine how languages fit some requirements. These requirements should cover the different phases of program development i.e. specification, design, implementation and verification.

The underlying metaphor of object orientation is that of largely autonomous objects with encapsulated states interacting by message passing. This has led many researchers to design concurrent object-oriented programming models. It is well known that concurrent programs require a much more careful analysis to prove them to be correct. We propose to adapt formal methods used for the specification and verification of conventional concurrent programs to an object-oriented framework. The usual definition of an object consists of an identity, a state, and methods which modify the state. A software system is composed of several objects which interact by means of messages. Methods are usually partial, i.e. they may be safely executed only if certain preconditions hold. In a distributed environment, the caller usually cannot guarantee that the precondition of the called method is met. Hence, preconditions are made explicit as enabling conditions, and messages may be blocked if the corresponding method is disabled. Our aim is to define a formalism for proving properties of agents and agent systems, reflecting their structural definition by inheritance and parallel composition. It therefore has to be compositional w.r.t. both these means of software construction. To make the model intuitive and avoid unnecessary complexity, parallelism occurs only between different agents, whereas each agent executes his methods strictly sequentially. All communication among the agents as well as between the environment and the agents occurs by explicit message passing. Therefore, we propose to distinguish between three levels of reasoning about systems built from concurrently executing agents, each focusing on one particular aspect or view of agent systems. The action level is concerned with the local effects of single methods or actions offered by an agent. The execution of a method or action transforms the agent's local state. Therefore, a simple formalism based on pre- and post-conditions of methods and actions is sufficient at this level. At the agent level, we reason about behaviors of individual agents of a given class, dealing with both safety and liveness properties. Proof rules take advantage of the encapsulation of an agent's private state which may only change by executing methods or actions defined for the agent. In contrast to the action level, initialization and fairness conditions are taken into account. Aspects of communication and interaction with other agents are not dealt with except in the form of environment assumptions. Finally, the system level models the top-level view of the entire system by an external observer. In particular, it cannot refer to the internal state of any object. Only the execution of actions by agents and the transmission of messages are visible. Typical properties of interest include synchronization and liveness involving several agents. We will define three different formal languages and proof rules, one for each aspect of reasoning. These languages will be related by ”interface rules”. Care will be taken to obtain simple and intuitive transitions from one level to the next. The conflict between inheritance and synchronization constraints is nothing else than a particular case of the more general problem raised when method constraints and inheritance are put together. Indeed, a synchronization constraint is a method constraint with a different semantics. Whereas a non-satisfied precondition raises an exception in the sequential case, it requires waiting in the parallel case. In our approach [BMS95], the problems of synchronization constraints caused by parallelism between methods disappears, since the agent itself executes the methods. Inheritance is treated at the sequential level, and all the aspects related to parallelism are treated at the composition level. The three levels of logic proposed correspond well to our intuition. The properties expressed with the logic of actions are automatically inherited (if, of course, the concerned method is not redefined or extended in the subclass). Therefore, only the proofs of agent properties have to be checked. But, under some conditions, we can reuse the existing proof of a property in a subclass. An environment to develop applications using this model has been implemented [BFS00]. On the one hand, this environment allows us to produce programs coded in Java from the model. On the other hand we have elaborated two decisions procedures to verify properties on the agent level and on the system level [BEBY99].

References

[BEBY99] Jean-Paul Bahsoun, Rami El Baida, and Hugues-Olivier Yar. Decision procedure for temporal logic of concurrent objects. In P. Amestoy and alt, editors, *Europar'99*, volume 1685 of *LNCS*, pages 1344–1352. Springer, 1999.

[BFS00] Jean-Paul Bahsoun, Pascal Fars, and Corinne Servires. *Object-Oriented Parallel and Distributed Programming*, chapter Multilevel Proof System for Concurrent Object-Oriented Systems, pages 31–52. Herms Science publications. Herms, 2000.

[BMS95] Jean-Paul Bahsoun, Stephan Mertz, and Corinne Servires. *Protocoles et programmation dans les rseaux*, volume 4 of *Paralllisme, rseaux et rpartition*, chapter A Framework for Programming and Proving Concurrent Objects, pages 65–98. Herms, 1995.