Incremental Object-Oriented Implementation of Concurrent Systems based on Prototyping of Formal Specifications

Didier Buchs, Jarle Hulaas

Swiss Federal Institute of Technology, Lausanne
Department of Computer Science
CH-1015 Lausanne
Switzerland

E-mail: buchs@di.epfl.ch or hulaas@di.epfl.ch

Keywords: Executable specifications, Mixed prototyping, Algebraic and Object based specifications, Abstract Data Types, Object-Oriented languages, Polymorphism.

1. Introduction

Formal specifications are taking an ever increasing importance in industrial applications. In particular for concurrent systems where behavior prediction and validation of the system under development are complex tasks. The advantage of formal methods resides by definition in their unambiguous nature. They can thus be supported by tools which automatically verify the properties, extract useful information or transform them into other representations. Moreover, they require a thorough analysis of the modelled system during the very first levels of the development process.

The benefits of formal specifications become even more evident when they can be used continuously through several stages of the software development cycle, like specification, implementation, integration or testing. Our purpose is to fill the gap that currently exists between the phase where a specification is simulated, generally using some logical inference tool, and the phase where the modelled system has an *efficient and maintainable* implementation in a main-stream object-oriented programming language. This objective is realized by application of a methodology called *mixed prototyping*.

Our work is done within the frame of the CO-OPN specification language [BG 91] and the associated development tool called SANDS [BFR 93&95]. CO-OPN (Concurrent Object-Oriented Petri Nets) is directed towards specification of complex distributed systems. The specification language is based on structured algebraic Petri nets [Rei 91], where the Petri nets are used to model operational and concurrent aspects, and the algebraic specifications [EM 85] serve as a means of abstractly describing the data types involved. The object notion serves as a structuring means, and naturally capture the concurrent aspects of a modelling. In this paper, we consider the way we can build incremental prototypes for concurrent systems based on object oriented language for CO-OPN. We made this experience with the object oriented part of the Ada 95 language [Ada 95](with PVM routines) but the principles can be transposed to any modern object oriented programming language. We use a minimal set of mechanisms in order to be largely language-independent and to make the prototyping process as intuitive as possible. (See for instance [Weg 87] for an explanation of object-oriented terminology). The basic principles can probably be generalized to any development method involving formal specifications. In a first part we explain the advantage of using prototypes and the general techniques that are used for building such prototypes. Then we explain in which sens, concurrency aspects can be incorporated in such framework. Finnally we give some indication to the mixed prototyping for CO-OPN.

1.1. Prototyping

Prototyping can be applied with different objectives and at different levels of the development process:

- Determining the feasibility of the product.
- Evaluating of the realization cost.
- Establishing and validating the requirements.
- Validating the functional specifications.
- Supplying information about the performance of the system, an evaluation of execution costs.
- Serving as a model, or a dynamic representation of it, especially during system testing.

Starting from the initial requirements, there are mainly three possibilities for using prototyping in the software development process:

- *Exploratory* programming is the most primitive direction. It is very close to traditional application development, in that there is no specification: modules are added to the system one after another. This approach may lead to unexpected results when integrating a new module because of its side-effects (particularly in parallel environments).
- *Throw-away* prototyping, where the prototype mainly serves as a way of validating the requirements - or part of them -, after which the operational system has to be constructed from scratch.
- *Incremental* prototyping, where the prototype is developed through various steps leading to the real end-user application. This implies that the prototyping tool provides for acceptable characteristics concerning completeness, speed and robustness.

Our proposition, *mixed prototyping* is a methodology for incrementally replacing abstract modules, i.e. automatically generated, with concrete ones, i.e. hand-coded. This can be done while continuously preserving the possibility of testing the modified prototype: Figure 1 illustrates a hierarchy of modules under development where some modules are abstract, some completely concrete, and some others in an intermediate state. This situation still allows to execute the prototype, provided that it is in a minimally coherent state. The concept of mixed prototyping was first introduced by Choppy [Ch 87]

Compared to the family of tools producing prototypes, this idea is certainly closest to the kind which generates procedural languages, because the abstract code is not meant to be read or modified by humans. Rather, the intent is to give the opportunity to add pieces of software that are automatically and harmoniously incorporated into the environment. This means that the automatically generated modules can take advantage of all the low-level optimization techniques developed for compiling ADTs and objects, provided that they are compatible with the general implementation protocol which ensures a clean interaction between the abstract and the concrete modules.

Other benefits of mixed prototyping are:

- No need for stubs or driver routines for the interconnection of abstract and concrete code, since they are written in the same language and are designed to cooperate following a unique protocol.
- The previous result also reduces the need for integration tests.
- Mixed prototyping supports both top-down and bottom-up development schemes.
- It encourages systematic refinement schemes.
- Reuse of concrete basic modules brings instant efficiency in new developments.

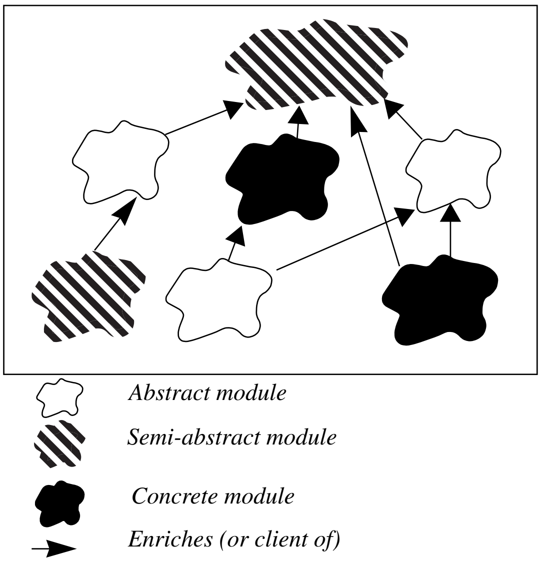

FIGURE 1. The concept of mixed prototyping

The correctness of partial implementation of models, which this methodology is based on, was proved by Choppy and Kaplan in [CK 90][CKM 89].

We apply these results to the frame of object-oriented languages. From now on we will talk of *abstract* classes, *symbolic* classes, and *concrete* classes (Figure 4 depicts this organization). Use of object-orientation adapts advantageously the principles of mixed prototyping to modern programming languages and paradigms. Compared to previous results, we can state the following additional improvements:

- The programmer autonomously redefines and specializes operations of the abstract class. The prototyping tool does not need to know what changes have been done, only which class to use as implementation.
- There is no hidden mechanism: We exploit the natural resources of the object-oriented paradigm.

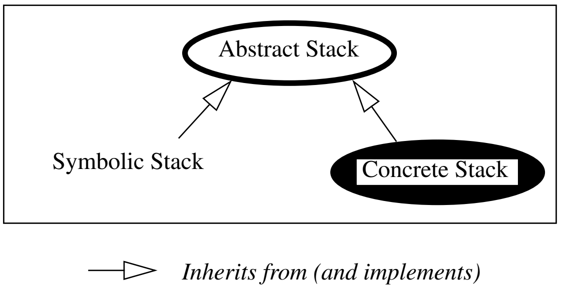

FIGURE 2. Object-oriented view of mixed prototyping (Stack exemple)

The prototyping process gains transparency.

- Client classes need not be modified nor even re-compiled when the developer switches forth - and even possibly back - between the symbolic and the concrete classes: the methodology reduces the impact of change.
- Memory management is transparent to the clients. The abstract class can even totally discharge its derived classes of this burden.
- Heterogeneous distributed systems are addressed by additional input-output facilities.
- Finally, as a primitive way of checking the correctness of the concrete code, we show how to add assertion and comparison mechanisms through automatically generated sub-classes.

By decoupling the client from the provider we gain the dynamicity and the flexibility that is needed for a prototyping method to be attractive.

1.2. Implementation model

There are two categories of execution techniques for prototypes based on Petri nets: *centralized* and *distributed*. The centralized approach is rather based on a simulation of the associated net, and is closer connected to interpretation techniques than to compiling. We have opted for the distributed mode, which is the most natural and efficient way to represent a Petri net. This does not exclude the existence of a central resource manager.

Coarse grain parallelism is needed for the execution of each CO-OPN object, an object being a sub-net of arbitrary size defined by the programmer. These units evolve naturally in a loosely coupled fashion. Synchronisation between objects and parameters of methods are implemented with message passing. As transitions in Petri nets are considered instantaneous, complex internal synchronisation mechanisms are needed.

The abstract machine then has to be mapped by the compiler onto a concrete machine, ranging from heterogeneous networks of workstations to massively parallel supercomputers.

1.3. Concurrency and mixed prototyping

When considering specification languages including concurrent objects such as CO-OPN, it is necessary to include other considerations about the development of mixed prototypes. Following the well admited idea that concurrency control and object functionnalities must be managed differently within concurrent object language, we split each concurent objects into two classes called *Control* and *Action* (see Figure 3).

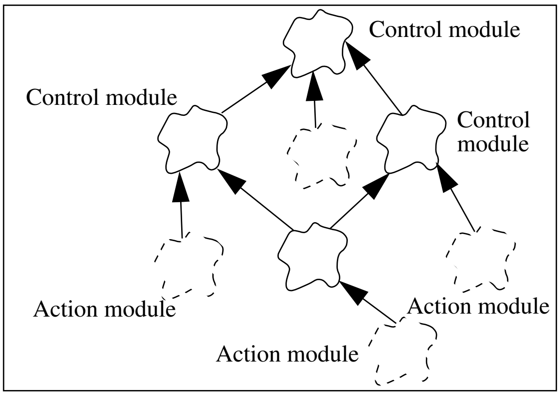

FIGURE 3. Hierarchy in mixed prototyping for concurrency

Control classes can be implemented through different algorithm depending on the kind of actions that must be supported (Figure 4).

- *Full control* implements the complete control scheme available in CO-OPN, this made by adopting a three phase distributed algorithm taken into account all entities implied in the dependancy hierarchy of the object under control and guarantying the atomicity of the synchronous comunication of the object. Eventual simultaneous synchronisation over shared object is also delegated to sub object in the hierarchy. It must be recalled that CO-OPN includes an abstract synchronisation operator which allow to express properties about true concurrency and simultaneous use of ressources. The phases are repectively the locking phase, the transmission phase with the corresponding reception phase and the unlocking phase.
- *Simple control* is a reduced version of the previous

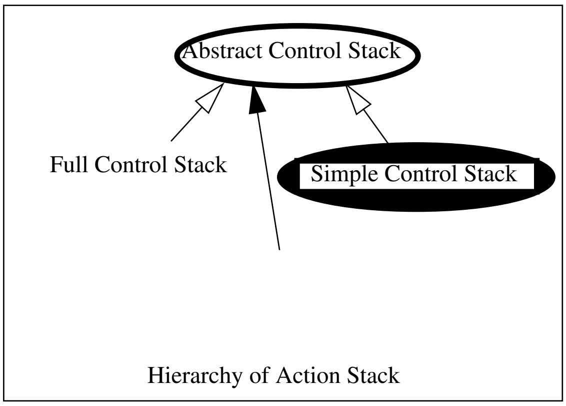

FIGURE 4. Object-oriented view of mixed prototyping for concurrency(exemple of Stack)

one where simultaneity is not taken into account. This can be enough when dealing with systems of objects adopting a tree structure. This is the case of one CO-OPN specification obtained after the application, in the environment SANDS, of divided to conquer transformation in the process of parallelization.

2. The mixed prototyping process

We can formulate the development scheme advocated for mixed prototyping.

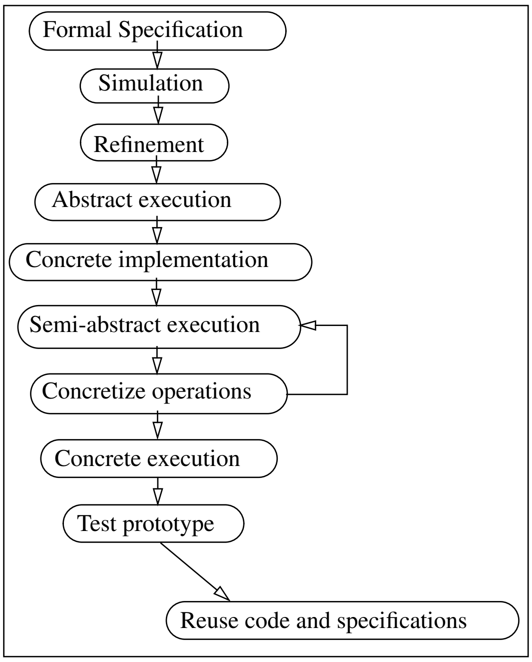

FIGURE 5. The mixed prototyping process

This illustrates the fact that the prototype can be executed at any moment. The benefits of object-orientation do not directly show on the Figure 5: it provides a flexible and intuitive approach. Moreover, it gives better guarantees for correctness.

The mapping process has two inputs: the specifications, expressed with the primitives of the abstract machine, and a description of the target execution platform, consisting of the number of available processors and its communication network topology and efficiency. Machine-specific hardware may be used to gain performance: for instance, the shared memory mechanism and the fast communications support of the Cray T3D . These will allow implementation of finer grained tasks as mentioned above.

3. Results

We shortly sketch the principles that are applied to build incrementally prototypes from abstract specification. It seems that the importance of the results concerns principally the domain of the software engineering for concurrent systems, in particular the improvement of the software development process. By adopting our methodology, users can considerably improve the global quality of the system under development. Nevertheless, several questions remains open, we can mention the evaluation of the real performance that can be expected by a prototype comparing to a hand written application. The user acceptance is also difficult to evaluate, because we disturb in a certain sens, the normal use of object-orientation.

4. Conclusion

In this paper we have shown how object-orientation gives a new dimension to mixed prototyping in particular that concurrent aspects can be included in this framework: It is more intuitive because it uses the natural resources of object-oriented programming. It brings more flexibility, allows a faster modify-compile-execute cycle, and gives better guarantees for the correctness of the concrete code.

5. References

[Ada 95] *Ada Reference Manual, Language and Standard Libraries,* ISO/IEC 8652:1995

[BG 91] D. Buchs, N. Guelfi, *CO-OPN: A Concurrent Object Oriented Petri Net Approach*, 12th International Conference on theory and application of Petri Nets, Aarhus, pp. 432-454, 1991

[BFR 93& 95] D. Buchs, J. Flumet, P. Racloz, *SANDS Structured Algebraic Net Development System,* Research report no 71, CUI 1993, also in 14th Int. Conf. Th. Petri Nets, Tool Presentation, Chicago, 1993. also 16th Petri Nets 1995, Turin.

[Ch 87] C. Choppy, *Formal Specifications, Prototyping and Integration Tests*, Proc. of the 1st European Software Engineering Conference, 1987

[CK 90] C. Choppy, S.Kaplan, *Mixing Abstract and Concrete Modules: Specification, Development and Prototyping,* Proc. 12th Int. Conf. on Software Engineering, Nice, 1990

[CKM 89] C. Choppy, S. Kaplan, V. Meissonnier, *Mixing executable specifications and code evaluation: An approach for prototyping and integration testing,* Proc. International Congress on New Technologies for Software and Supercomputers Development, Caracas, November 1989

[EM 85] H. Ehrig and B. Mahr , *Fundamentals of algebraic specification 1: equations and initial semantics,* EATCS Monographs, Springler Verlag, 1985

[Rei 91] W. Reisig, Petri Nets and Algebraic Specifications, Theoretical Computer Science 80, pp. 1-34, 1991

[Weg 87] P. Wegner, *Dimensions of Object-Based Language Design,* in OOPSLA '87 Conference Proceedings, pp. 168-182