OOCP = OOP + C
Position Paper

Jean-Pierre BRIOT

Equipe Mixte LITP - Rank Xerox France,
Université Pierre et Marie Curie,
4 place Jussieu, 75005 Paris, France
briot@rxf.univ-p6-7.fr

Abstract

This position paper discusses the following statement: "Object-Oriented Programming (OOP) is currently the best paradigm to describe Object-Oriented Concurrent Programming (OOCP)". Three aspects are considered: implementation, description ability, and programming environment. The Actalk system (which stands for actors in Smalltalk) is used along the discussion as an example supporting our position statement.

Keywords

object, concurrency, actor, implementation, message, description, classification, class, modularity, inheritance, genericity, visualization, programming environment, Smalltalk-80, Actalk

1 Introduction

The position we defend in this paper is:

"Object-Oriented Programming (OOP) is currently the best paradigm to describe Object-Oriented Concurrent Programming (OOCP)"

We summarized this statement in the title "equation":

"OOCP = OOP + Concurrency"

If we take a reflective approach, OOCP itself would be the best candidate to describe OOCP. Reflection is a very promising research direction and is now better understood in the field of OOP. However it is still in early infancy in the field of OOCP, the most advanced proposal being the reflective description of the ABCL/1 language by Watanabe and Yonezawa [Watanabe and Yonezawa 89]. However we left aside this approach, at least now. We will rather restrict the complete OOCP model to some subset: (sequential) OOP. Because OOP is yet much more developped than OOCP, we gain efficient and sophisticated environments to design, classify and visualize OOCP.

A system, named Actalk, will be used along this position paper to examplify and validate our position. Actalk uses descriptive and visual ability of the Smalltalk-80 [1] environment to provide a testbed for designing and classifying actor-based languages. Actor-based languages form one of the most active and open family of OOCP languages. For this reason the goal of Actalk was not to define a new OBCP language, extension of Smalltalk-80 towards concurrency, like for instance ConcurrentSmalltalk of Yokote and Tokoro [OOCP 87, pages 129–158]. Actalk provides an open testbed to model various OOCP languages and to experiment with them in a single environment.

[1] Smalltalk-80 is a trademark of ParcPlace Systems.

2 Advantages of OOP to Describe OOCP

The merits of choosing OOP as a basis to describe OOCP will be now discussed along three successive points:

- At first we will discuss the implementation support. We will review why OOP gives for free the best support for OOCP, in terms of minimality and modularity.
- The second point is about the descriptive ability of OOP. We will show that classes, inheritance and genericity are very useful tools to describe OOCP.
- We will complete our survey by discussing the gain of reusing the integrated and modular programming environment of Smalltalk-80. This underlying environment will be used to prototype environments suitable for OOCP.

3 Implementation Ease

3.1 Minimality for Pedagogy

OOCP languages introduce the notion of active and concurrent objects. Smalltalk-80 provides all entities needed to build them: objects, classes and messages, and to express concurrency: processes and semaphores. Thus implementation is high level and minimal. It may focus on the semantics of OOCP and the differences with standard OOP, rather than deal with low level concern.

3.2 Combination for Reuse

Actalk models OOCP by defining a sub-world of active objects embedded into the world of standard objects. This leads to a modular implementation and a very good combination between passive and active objects. This results in the maximal reuse of the underlying system.

3.3 Modularity for a Layered Architecture

The kernel of the Actalk system which describes the minimal semantics of active objects is defined by two simple classes. The first class, named `Actor`, defines the semantics of active objects (actors), and the class `ActorBehavior` defines the semantics of their behaviors, which describe how messages are interpreted. The class `Actor` simply defines an actor through its two components, the mailbox or queue which will contain the incoming messages, and the behavior which will process them. The class `ActorBehavior` expresses the default semantics of the activity of behaviors, i.e., keep dequeueing and performing messages from the mailbox. Both classes may be extended by using inheritance in order to simulate various OOCP models.

4 Descriptive Abilities

4.1 Classes for Abstraction

Classes of behaviors of actors are implemented by standard abstractions available in OOP, i.e., classes. Consequently, the language designer and the programmer may use the standard Smalltalk-80 classification system and environment (browser) to define new OOCP models and programs.

4.2 Inheritance for Classification

In order to simulate some of the most representative OOCP computation models and programming languages based on the actor concept, we use the main classification tool of OOP, i.e., inheritance. Various models will be described as progressive extensions from the primitive kernel.

A first extension simulates the Actor computation model of Agha and Hewitt [OOCP 87, pages 37–53] as a subclass of class `ActorBehavior`, named `AghaActorBehavior`. Only two methods need to be defined in class `AghaActorBehavior` to implement the concept of *behavior replacement*, i.e., how to process next incoming message.

The second example simulates the Abcl/1 programming language of Yonezawa [OOCP 87, pages 55–89] as a subclass of class `Actor`. This extension easily implements three distinct types of communication protocols between actors: asynchronous, synchronous and eager types of message passing.

These simulations show the merits of modularity for our kernel. Because the kernel and its extensions are related by inheritance, one could easily compare them. Inheritance helps not only to classify various actor models, but also to clearly relate and to reuse their various implementations.

4.3 Genericity for Modular Control

Another extension of the Actalk kernel introduces a generic control of actor events (i.e., receiving a message, computing it...) into Actalk. This is implemented by defining generic event methods instantiated for various applications. We currently use this facility to design actor event driven representations (views) of actors, and to modularly change the semantics of the underlying Smalltalk-80 scheduler of processes.

5 Environment Support

5.1 Reuse of Standard Programming Environment

A good programming environment is important for prototyping and experimenting with sequential languages. It becomes almost necessary when experimenting with concurrent languages. However it is usually lacking because the task, specially for debugging, is not trivial. Smalltalk-80 is the most achieved and flexible OOP system with a fully integrated programming environment. Because of the integration of our actors into the Smalltalk-80 model and environment, the designer of actor languages and the programmer could automatically reuse the standard Smalltalk-80 programming environment.

5.2 Extension Towards a Specific Environment

However, the standard Smalltalk-80 programming environment cannot take into account some specificities of OOCP. For example, the debugger could not show useful information when asynchronous message passing is used. The Smalltalk-80 MVC (Model View Controller) paradigm for interface design makes stong hypotheses about the sequentiality of the language. Therefore reframing a view on some active object which updates itself may cause to chaotic redisplay.

For this reason it is necessary to extend the standard environment towards a specific one. By starting from the standard programming environment and by using inheritance to extend it, we may quickly design such specific environments. The Actalk project includes such a prototype extension including specific debugger and extended MVC system.

6 Conclusion

In this paper we advocated the use of current OOP support, precisely the Smalltalk-80 system, to design and experiment with OOCP programming. We believe this is a promising approach both for designers wishing to prototype and users wishing to study and experiment with OOCP.

References

[Briot 89] J.-P. Briot, Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment, European Conference on Object-Oriented Programming *(ECOOP'89)*, to be published by Cambridge University Press, U.K., July 1989.

[OOCP 87] Object-Oriented Concurrent Programming, edited by A. Yonezawa and M. Tokoro, *Computer Systems Series*, MIT Press, Cambridge MA, USA, 1987.

[Watanabe and Yonezawa 89] T. Watanabe and A. Yonezawa, Reflection in an Object-Oriented Concurrent Language, Conference on Object-Oriented Programming Systems, Languages and Applications *(OOPSLA'88)*, Special Issue of SIGPLAN Notices, ACM, pages 306-315, Vol. 23, No 11, November 1988.