INTENSIONAL COMMUNITIES

JOHN PLAICE
School of Computer Science and Engineering,
The University of New South Wales, Sydney 2052 Australia.
E-mail: plaice@cse.unsw.edu.au

PETER G. KROPF
Département d'informatique, Université Laval,
Québec (Québec) Canada G1K 7P4.
E-mail: kropf@ift.ulaval.ca

We describe the interaction and relation between entities in distributed systems, as proposed in the Web Operating System (WOS). Every entity in the system is a versioned object which depends on its current context, which itself is programmable and can be effected by the objects circulating within it. These entities interact through mechanisms of requests/answers and negotiations. Those who exhibit functional and behavioral affinities may dynamically associate themselves to form communities. This positional paper states the basic ideas of the notion of communities in distributed systems.

1 From OO to Intensional Communities

With the emergence of widespread computing and telecommunication networks, there has been an explosion of networked and mobile computing. A permanent increase in bandwidth drives the growth in multimedia and mobile services, in electronic-commerce, in large-scale high-performance distributed computing, as well as in the number of Internet users. In general, it can be said that the global computing infrastructure is in a permanent process of *evolution*.

Because of the rapid changes in the underlying infrastructure, it is widely acknowledged that component-based systems are best suited for large-scale distributed systems, since, as needs change, components can be replaced more easily than can entire systems. Like Meyer and Mingins,[1] we consider the theoretical model for components to be object-oriented technology.

If one of the problems in developing distributed systems is the changing underlying infrastructure, i.e. a changing context, then it becomes interesting to examine methods for *versioned programming*, where components can potentially act differently according to the context in which they are immersed.

This goal of versioned programming is finally becoming viable with the development of Swoboda and Wadge's ISE,[2] a Perl-like scripting language in

which one can program several versions of functions, and change as needed the context in which these functions are to execute.

ISE is the most recent development in a long line of languages and tools that have put forward the concepts of *intensional programming*, where programs are understood to execute in an implicit multi-dimensionsl context, whose individual dimensions can be manipulated explicitly as needed by a program.[3]

What distinguishes existing work in intensional programming from distributed computing is that the existing work concerns itself only with a single object or program, immersed in a manipulable context.

However, distributed computing means that *multiple* objects, arising from several sources, will be put together in a single context; furthermore, objects will not necessarily remain in a given context, and may migrate to other contexts. We therefore have a potentially much richer model for intensional programming.

One must immediately ask if the model — clearly not developed — presented in the previous paragraph is useful. In fact, it is much more powerful than one might imagine.

At its purest level, object-oriented technology is simply *atomism* applied to computing: everything is an object, and objects communicate through message-passing, which means a message is sent from one object to another through some sort of rendez-vous process. *There is no context; outside of the objects there is nothing.*

But distributed computing is *intuitively* understood as supporting the creation of communities. For example, Sun's Jini connection technology is presented as supporting the creation of *federations* of components.[4]

But a community is *not* simply the juxtaposition of a set of objects. Noone would claim that putting together in the same room a number of people implies creating a community. Rather, a community, at whatever level, consists not only of a number of people but, also, a number of ties that implicitly or explicitly link these people together: a workplace, shared interests, membership in a club, a religious or national grouping, etc. Of course, these links, with their implicit knowledge, familial ties, rules, obligations, are subject to continuous negotiation within each community.

When we think of communities, we should understand that there are many different kinds of community. Some are public, some allow loose affiliation, others are for very restricted, closed circles. Others only exist for very short periods. In fact, the diversity of chat rooms on the Internet should give some idea of the different kinds of communities that can arise. In addition, different communities may agree to trade, meet, or discuss, depending on

the communication protocols that they can agree upon. Other examples of communities are the intranets of large organizations.

The same should hold true when objects are brought together. We should be able to program the equivalent of the ties that bind together communities. In other words, we should be able to have *explicit* contexts that can be manipulated by the objects that belong to them.

The point is not simply to make an object model that is more in keeping with real life. Rather, we wish to create a *much more powerful* programming model, one that allows *real* evolution, in which there is an interplay between the actions of the objects and of the communities in which they are placed. In this view, evolution takes place through the transformation of the environment by the objects it includes, as well as by the changes forced upon the objects by a chaning environment,

It should be understood that the objects are understood to be versionable, and in a certain sense, are akin to ISE programs. In addition, when an object manipulates its context, it is essentially broadcasting to the other objects within its context, and possibly forcing changes upon them. Broadcasting also serves as a bootstrapping tool, allowing an object to find communities that it may join.

The conceptual model as presented above is clear, and it should not be difficult to develop a generalization of the ISE language that supports communities. The contexts can be programmed through the use of threads or processes, or other means, so long as the different objects sharing the context can all be both readers and writers.

However, it is still unclear what the correct primitives for such a language should be. In addition, the different kinds of community processing also need to be examined. Furthermore, the programming model as defined above requires an infrastructure that supports versioned resource management and communication between versioned objects; the goal of the Web Operating System (WOS) project is precisely to create such an infrastructure.[5]

References

1. B. Meyer and C. Mingins. Component-based development: From buzz to spark. *IEEE Computer*, 32(7):35–37, 1999.
2. P. Swoboda and W. W. Wadge. Vmake, ISE and IRCS: General tools for the intensionalization of software systems. In *Intensional Programming II*. World-Scientific, Singapore, 1999. This volume.
3. J. Plaice and J. Paquet. Introduction to intensional programming. In *Intensional Programming I*, pages 1–14. World Scientific, Singapore, 1995.

4. Sun Microsystems. Jini. http://java.sun.com/products/jini/.
5. http://www.wos-community.org.