Published in Distributed Computing on the Web (DCW'99), 163-167, 1999
which should be used for any reference to this work

WOS COMMUNITIES — INTERACTIONS AND RELATIONS BETWEEN ENTITIES IN DISTRIBUTED SYSTEMS

John Plaice
School of Computer Science and Engineering
The University of New South Wales, Sydney 2052 Australia.
Email: plaice@cse.unsw.edu.au

Peter G. Kropf
Département d'informatique
Université Laval, Québec (Québec) Canada G1K 7P4.
Email: kropf@ift.ulaval.ca

ABSTRACT

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.

KEY WORDS

Web Operating System, versioning, eduction, communities in global computing.

INTRODUCTION

The WOS approach to global computing first presented in reference [2] aims to provide solutions for global ubiquitous computing and to develop service mechanisms which meet the requirements of the net-centric view of services and processes. To account for the dynamic nature of the Internet, generalized software configuration techniques, based on a demand-driven technique called *eduction* [6], are developed for the WOS. The kernel of a WOS node is a general *eduction engine*, a reactive system responding to requests from users or other eduction engines. A WOS-node integrates thus client, server, and broker/trader functions. These nodes are capable of providing a set of services, that can pass on to each other requests when appropriate. Because the Internet or the Web is dynamically changing in many directions, any attempt to design one single operating system offering a fixed set of resource management functions is fore-doomed. Therefore, the WOS is designed not only as a *distributed*, but also as a *versioned* system. Different versions of services and of the WOS itself are running simultaneously on the network. Warehouses associated with the WOS nodes provide the necessary information and components for fulfilling service requests. Each node thus uses its warehouses to store and continuously update information about the node and available services and resources. This approach allows for interaction with many different warehouses, each offering different versions of available services, resource management techniques, applications, platforms, hardware, and so on.

The WOS framework [4] consists in the sum total of the interactions between nodes relying on the basic communication protocol and multiple warehouses on each node. It is designed as a

completely open system to every user. Conceptually, the totality of WOS nodes constitute the *WOSNet* or *WOSspace*. However, since the WOS is a versioned system, subnets can be easily defined: a collection of some WOS nodes may be defined as a particular version of the WOSNet. For example, a number of servers of an Intranet or an Extranet could be defined as a WOSspace, i.e. a version of the WOSNet including only those servers. Every WOS compliant system can potentially contact those servers, but only if they are version-compatible.

There are several approaches to integrate the computational resources available over the Internet into a global computing resource and service infrastructure. The closest one to the WOS is the Jini architecture proposed by SUN Microsystems [5]. Jini allows one to build federations of nodes or distributed objects offering different services each relying on its own service protocol. Lookup services provide location and discovery functions. These services reside on some nodes and act as brokers in the federation.

The WOS approach is qualitatively different and more general in that federations, i.e. subsets of WOS nodes, defining a specific environment and context are dynamically and autonomously created. These subsets of interacting nodes who continuously define and redefine the relationships between them as well as their environment form **communities**. This is achieved with versioning and powerful lookup/discovery protocols and generalized service communication protocols.

Furthermore, as we shall see, WOS federations of nodes are not simply the juxtasposition of nodes, but the creation of real communities.

VERSIONS

Object-oriented programming considers every entity or component as an object. Pairs of objects may communicate directly or indirectly through a broker. Relations are always defined between a pair of components sitting in empty space. The object hierarchy and inheritance properties do not alter this basic behavior. Communities however, are much more than just side-by-side placement of isolated entities. An entity may exhibit different functional properties and behaviors depending on its context or the community it currently belongs to. In order to deal with this situation, we consider every component as a version defining the components current properties in a continuously changing environment. Any entity, be it a resource, a communication protocol, a service or a program is thus versioned. Moreover, in contrast to object-oriented programming, we propose to rather manage diversity and internal behavior than to hide them.

Version management in the WOS has been informally presented in [3]. It follows from previous work on software versioning [7] and intensional programming [6], in particular the Lucid programming language [1]. An intensional programming system is characterized by two operations, *lookup* and *eval* which define interactions with possibly multiple warehouses and definition catalogs. These catalogs are basically tagged objects denoting (context,value) pairs which represent identifiers. The two operations will, in general, have side effects, i.e. cause explicit changes of state by updating warehouses and catalogs along the way. The generalization and application of these concepts to the WOS allows thus for dynamically changing contexts and even creating new contexts of computation.

COMMUNITIES

When we refer to communities, we should first understand what they are not. The Jini architecture [5] allows one to build federations of objects; but that is not a community, in any sense of the word. No one would claim that by putting together in the same room a number of people, that would create a community. Rather, a community, at whatever level, consists not only of a number of people but, also, a number of links 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.

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

This debate is not a new one. The philosophical basis for object-oriented computing comes from the ancient Greek Demokritos (Democritus), who proposed that the cosmos is composed of indivisible atoms colliding in empty space.

Demokritos's vision was openly opposed by Aristoteles (Aristotle), who put forward that the atoms were (1) not indivisible; (2) nor did they circulate in empty space. Rather, at any level, atoms could always be divided again, and everything was permeated by and circulating in the *aether*.[1]

The WOS vision of objects is much closer to Aristoteles's vision of atoms than to Demokritos's. Our objects are not immutable, defining a fixed set of methods; rather, their functionality depends on the context in which they are placed. Furthermore, they are not necessarily indivisible: in certain situations, they may allow themselves to be opened, with some of their "components" being taken out and repackaged with the "components" of other objects to form new objects. Finally, objects placed within a context may interact, indirectly, through that context. Communication needs no longer take place explicitly through the channels provided by the objects themselves.

In the WOS universe, all objects are tagged with version information. One simple way to understand tags is as n-tuples of *attribute-value* pairs, where the attributes and values can be any ground data values. Contexts are also tags, but they are not necessarily ground. They allow for a large range of variation, and objects can circulate through a given context if their allowed variation is consistent with the allowed variation of the context. Should one restrict the other, then negotiation might have to take place with the other members of the community.

In addition, once objects become living members of a particular community, they can express themselves, through their methods, and their protocols, in a way particular to a given community. This comes from the fact that their very behavior is itself versioned, and depends on the current context.

CONCLUSIONS

The WOS notion of community and of mutable, adaptable objects that are flowing through explicit contexts is a very powerful idea that requires further development. The authors are currently working on the formalization of these ideas.

[1]This debate is best known with respect to the nature of light: does it consist of particles (Newton, Einstein, Schrödinger, ...), or waves (Huyghens, Maxwell, Lorentz, Planck, ...). We leave it to the reader to surmise the authors' opinion.

REFERENCES

[1] E. Ashcroft, A. Faustini, R. Jagannathan, and W. Wadge. *Multidimensional Programming*. Oxford University Press, New York / Oxford, 1995.

[2] S. Ben Lamine, P.G. Kropf, and J. Plaice. Problems of Computing on the Web. In A. Tentner, editor, *High Performance Computing Symposium 97*, pages 296 – 301, Atlanta, GA, April 1997. The Society of Computer Simulation International.

[3] S. Ben Lamine and J. Plaice. Simultaneous multiple versions — the key to the WOS. In *Distributed Computing on the Web (DCW'98)*, pages 122–128, Rostock, Germany, June 1998.

[4] Peter Kropf. Overview of the WOS^{TM} project. In *1999 Advanced Simulation Technologies Conference (ASTC 1999)*, San Diego, CA, USA, April 1999.

[5] Sun Microsystems. Jini. `http://java.sun.com/products/jini/whitepapers/`.

[6] Joey Paquet. *Intensional Scientific Programming*. PhD thesis, Faculté des études supérieures, Université Laval, Québec, Canada, 1999.

[7] J. Plaice and W.W. Wadge. A new approach to version control. *IEEE Transactions of Software Engineering*, 19(3):268–276, 1993.