# A Panacea for Distributed CSCW Infrastructure?

Andrew Berry
Department of Computer Science
The University of Queensland
<andyb@dstc.edu.au>

## 1 Introduction

One of the major difficulties of building and deploying a distributed CSCW system is choosing or building an appropriate distributed infrastructure. While there are a number of solid infrastructures for building distributed information systems, most of these fall a long way short of providing the flexible and dynamic support required by CSCW systems.

Research and commercial development of distributed infrastructure has typically focused on support for building reliable information systems, for example CORBA, DCE and Isis/Horus[14]. These infrastructures provide programmer-oriented functionality in the form of remote procedure call or multicast and unicast messaging. These interaction mechanisms are relatively low-level and static, requiring significant additional programming to support the higher-level interaction mechanisms found in CSCW systems. The infrastructures also tend to be closed, in that all application components[1] must be written specifically for the distributed infrastructure being used.

Existing CSCW systems, for example wOrlds[5] or TeamRooms[6] have resorted to building their own distributed infrastructure services by extending existing services and building new infrastructure to suit their immediate needs. While these approaches have been effective, considerable developer effort has been expended in developing and configuring necessary infrastructure. Since these are prototypes whose primary goal is to demonstrate and support CSCW research, the resulting distributed infrastructure is often inflexible and can be difficult to deploy or extend.

My thesis is that the effort required to build, configure and deploy distributed infrastructure for CSCW systems can be significantly reduced by the use of an executable *architecture description language*. Such a language would support specification of the configuration of distributed application components and the interactions of these components in a manner independent of the underlying network and protocols. By providing an execution engine for the language, CSCW application developers need only provide a description of the configuration and interactions of their application components. The development of an architecture description language and underlying distributed infrastructure support is currently being pursued at the University of Queensland in conjunction with the CRC for Distributed Systems Technology (DSTC). This position statement outlines the plan and progress to date of this work.

## 2 Plan and Progress

### 2.1 An Architecture Description Language

Recent research in the software engineering discipline has found that in large software systems, the configuration and interactions of the components is as complex as the software components themselves. This aspect of a software system is traditionally called the *architecture* of the system and has been treated

[1] The term "components" in this paper refers to executable software components. The term "object" is avoided because it tends to be ambiguous across disciplines.

informally or semi-formally in the design process. Software developers typically sketch the architecture with boxes and lines, relying on the discussion around the sketch to define the semantics. When it comes to coding the software components, the architecture is usually embedded in the components, which makes them less flexible and more difficult to re-use. To address these problems, researchers have begun modelling the architecture of a software system in a rigorous and semantically well-defined manner. A comprehensive review of work in this area is provided by Shaw and Garlan in [13].

A number of approaches are being used to define software architecture, including graphical tools with well-defined semantics and architecture description languages (for software, as opposed to hardware architecture description languages). These approaches are intended to remove configuration and interaction code from components, both supporting greater re-use of components, and capturing the architecture of a solution for documentation and potential re-use.

In this work, we have chosen to implement an *executable* architecture description language as a basis for description of component configuration and interactions. By *executable*, we mean that it will be compiled or interpreted and run over a suitable distributed systems infrastructure. The key to the success of the proposed work is development of a practical, programmer-friendly language for describing the architecture of distributed CSCW systems and applications. This work is well underway, and is driven by two areas of research:

1. Work on software architecture and connection languages, in particular the Rapide system[9] and the work of the Software Engineering Institute at Carnegie Mellon University[1][12].
2. Ongoing work on distributed systems architecture at the DSTC, and in particular, the A1! Architecture Model[4] and the Reference Model of Open Distributed Processing (RM-ODP)[7], an ISO standard reference model for distributed systems architecture.

These aspects are largely complementary, with the first aimed at providing flexible support for describing software architectures and the second aimed at mapping such software architectures onto a distributed infrastructure. Our recent work has been the development of a semantic model for architecture description languages suitable for distributed infrastructure[10] based on the following fundamental constructs:

**interface:** software components have interfaces through which they interact with their environment (via bindings).

**binding:** a binding is an infrastructure-provided configuration of network connections and behaviour. A binding specification describes a configuration of components and their allowed or expected interactions. Components connect to a binding through their interfaces and must satisfy the expectations of the binding specification.

**events:** components participate in a binding (interact) by executing events at their interface. Events have parameters and direction (in or out). Events may be non-atomic, that is, they can be built from a configuration of lower-level events.

**event relationships:** event relationships specify the behaviour and interactions of a binding by describing the relationships between events occurring at component interfaces. An event relationship specification describes:

- the ordering of events, which is a partial order because the components are distributed and operate in parallel,
- the relationships of event parameters, with arbitrary functional relationships permitted between related events, and
- real-time constraints, for example, specification of the maximum allowable delay between the emission of an event and a corresponding reception of a related event.

Interactions such as remote procedure call and multicast are described as an event relationship. In our architecture description language, abstraction and encapsulation facilities will allow pre-defined descriptions of these and other common interactions.

The arbitrary functional relationships between parameters are extremely powerful, allowing the description and implementation of bindings that can, for example, connect DCE components to CORBA components. The potential for unbounded complexity resulting from arbitrary functional relationships is dealt with by allowing a binding implementation to restrict the available functions.

The semantic model also provides highly-flexible facilities for abstraction, allowing us to hide low-level, protocol-oriented behaviour and provide a concise, high-level view of (abstract) events that are relevant to the application or interaction being described. This could be used, for example, to describe video communication in a manner that is independent of its implementation, yet retain compatibility with a particular implementation.

Work is currently underway to use this semantic model as the basis of an architecture description language. This language will describe binding and interface behaviour using events and event relationships. In addition to the core elements of the semantic model described above, it is intended that the language provide encapsulation, implementation inheritance, and dynamic modification of behaviour through access to a meta-level. These facilities are intended to give a structured language with the flexibility necessary to describe CSCW applications.

### 2.2 Investigation and Description of CSCW Systems

Our work on architectural models for distributed systems has been in progress for several years. It has become clear that for the work to be useful, it must adequately support a wide variety of applications and interaction styles. To validate and refine the model and infrastructure, CSCW systems have been chosen as a target application area due to the inherent complexity of interactions and the need for flexible and dynamic infrastructure support.

In parallel with the architecture description language work, this author has begun to gather the of CSCW systems requirements through participation in an evaluation of wOrlds[5]. The following key requirements have been identified:

- It should be possible to incorporate existing application components and interaction mechanisms in a cohesive and flexible manner;
- The infrastructure should support dynamically configurable replication with a variety of mechanisms to deal with varying bandwidth, latency and coordination requirements in Internet-scale networks;
- It should be possible for users to dynamically vary their participation in and awareness of collaborative activities. This is necessary to support the ever-changing work focus of individuals and the varying quality of service provided by the network.

It is intended that other CSCW systems, literature and infrastructure toolkits, for example the relevant COMIC work[3] and GroupKit[11], will also be evaluated to develop a well-rounded set of requirements for distributed systems infrastructure. The author is also participating in the design of a successor to wOrlds known as Orbit[8], which will be the source of additional requirements.

As an evaluation of the language work described in 2.1, the interaction mechanisms provided or required by these systems will be specified in the resulting language. This will both ensure that the language is capable of describing such systems, and provide a basis for duplicating the infrastructure of those systems once the prototype execution environment has been completed.

### 2.3 Prototyping

The binding language described in section 2.1 requires an environment to support its execution. Given such an environment, a CSCW system developer could, for example, specify a suitable replication strategy in

a high-level language and have it implemented directly by the execution environment. This binding specification can be re-used in higher-level bindings that describe application-oriented features, for example, shared document editing.

A prototype distributed infrastructure supporting the principles of the A1! Architecture Model[4] has been developed at the DSTC. The Hector[2] prototype aims to provide support for connection between distributed components using arbitrary protocols based on a binding description. The focus at present is on the establishment of such connections in an open environment, including negotiation of binding specifications and parameters. In its current form, Hector provides only a relatively low-level API for software developers.

The goal of this author's work is that an interpreter for the binding language described in section 2.1 will execute within the Hector prototype. The researchers working on Hector are in close contact with the language work, ensuring that our respective goals and requirements remain synchronised. The result should be a language and environment for describing and implementing complex CSCW systems over a distributed infrastructure.

## 3 Discussion

The preceding sections outline the plan and progress of work intended to provide an infrastructure and language environment to support distributed CSCW applications. This work is dependent on the progress of the various pieces that make the puzzle, and leaves us with a number of interesting questions:

- Can a feasible distributed infrastructure adequately support the flexible and dynamic nature of CSCW applications?
- Is it possible to describe the necessary mechanisms of interaction using the semantic model that has been developed?
- Are architecture description languages acceptable to developers building CSCW applications and systems?

It is intended that these questions and others will be answered by the continuation of this work. A successful outcome will mean that many of the distributed infrastructure problems currently faced by CSCW researchers will cease to exist.

## References


[1] R. Allen and D. Garlan. Formalizing architectural connection. In *Proceedings 16th International Conference on Software Engineering*. IEEE, May 1994.

[2] D. Arnold and A. Bond. An interaction glue for middleware. DSTC Internal Report, Mar. 1995.

[3] S. Benford and J. Mariani, editors. *Requirements and Metaphors of Shared Interaction, COMIC Deliverable 4.1*. Esprit Basic Research Action 6225, Oct. 1993.

[4] A. Berry and K. Raymond. The A1! architecture model. In *Open Distributed Processing: Experiences with distributed environments*. IFIP, Chapman and Hall, Feb. 1995.

[5] G. Fitzpatrick, W. J. Tolone, and S. M. Kaplan. Work, locales and distributed social wOrlds. In *Proc. of the 4th European Conference on CSCW*. Kluwer Academic Publishers, 1995.

[6] S. Greenberg, C. Gutwin, M. Roseman, and A. Cockburn. From awareness to TeamRooms, GroupWeb and TurboTurtle: Eight snapshots of recent work in the GroupLab project. Technical Report 95/580/32, Dept. of Computer Science, University of Calgary, Dec. 1995.

[7] 10746-1 10756-2 10746-3 Basic Reference Model for Open Distributed Processing.

[8] S. Kaplan, G. Fitzpatrick, T. Mansfield, and W. J. Tolone. MUDdling through. To appear in IEEE Proceedings HICSS'97, 1996.

[9] D. C. Luckham and J. Vera. An event based architecture definition language. *IEEE Transactions on Software Engineering*, Sept. 1995.

[10] A. Rakotonirainy, A. Berry, S. Crawley, and Z. Milosevic. Describing open distributed systems: A foundation. To appear in IEEE Proceedings HICSS'97, 1996.

[11] M. Roseman and S. Greenberg. GroupKit: a groupware toolkit for building real-time conferencing application. In *Proc. 4rd Int. Conf. on CSCW*. ACM Press, Nov. 1992.

[12] M. Shaw. Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status. Technical Report CMU-CS-94-107, Software Engineering Institute, Carnegie Mellon Univerity, Jan. 1994.

[13] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an emerging discipline*. Prentice Hall, 1996.

[14] R. van Renesse, K. P. Birman, and S. Maffeis. Horus, a flexible group communication system. *Communications of the ACM*, Apr. 1996.