

ANTS Framework for Cooperative Work Environments

Designed to facilitate computer-supported cooperative work (CSCW), ANTS overcomes the major deficiencies found in earlier groupware toolkits at the conceptual, architectural, and technological levels.

Pedro García López

Universitat Rovira i Virgili

Antonio F. Gómez Skarmeta

Universidad de Murcia

The Internet's ubiquity, along with enormous growth in network bandwidth and computing power, has fostered a multitude of computer-supported cooperative work (CSCW) environments in recent years. These include instant messaging services such as ICQ and Yahoo Messenger, peer-to-peer file-sharing systems such as Napster and Gnutella, multiuser games such as Ultima Online, and shared workspace systems such as BSCW (basic support for cooperative work).

Key factors in facilitating the advent of CSCW environments include the use of open systems that permit system interoperability; advances in database, computer graphics, and display technology; and widespread acceptance of networking technologies and protocols. The open-ended communication possibilities of this emerging medium will soon lead to a boom in such environments, enabling remote users to interact in unprecedented ways.

GROUPWARE

Developing the software that runs CSCW systems, known as *groupware*, on conventional platforms presents a complex task that forces developers to repeatedly invent similar solutions to similar problems. Consequently, many R&D initiatives have embedded generic solutions into their groupware development platforms.¹ These platforms, or toolkits, seek to simplify groupware development by reusing solutions for the common problems found in multiuser collaborative applications.

Many groupware toolkits successfully abstract access to distributed services, thereby reducing the

learning curve. For example, the classic GroupKit² groupware platform provides programming abstractions such as multicast remote procedure calls, multiuser interface objects, session management events, and shared data structures—called *environments*—that have embedded collaborative consistency management.

Although powerful, GroupKit and many other existing platforms have three main deficiencies:

- At the *conceptual* level, they concentrate on multiuser services for groupware applications and thus do not define powerful monitoring and awareness services for analyzing the information the collaboration process generates.
- At the *architectural* level, they lack a consistent component model, which is necessary for developing collaborative third-party components that can be assembled into an application framework.
- At the *technological* level, they lack horizontal middleware integration and thus implement custom proprietary services, which siphon off development effort and resources to create the required middleware, precluding system interoperability and thus widespread acceptance.

Subsequent research assimilated results from the first groupware toolkits but emphasized component models to simplify development. Habanero Hablets,³ ComeCo Medium grouplets,⁴ and Disciple JavaBeans⁵ all define component platforms that considerably simplify development and software reuse. Unfortunately, these toolkits do not provide a com-

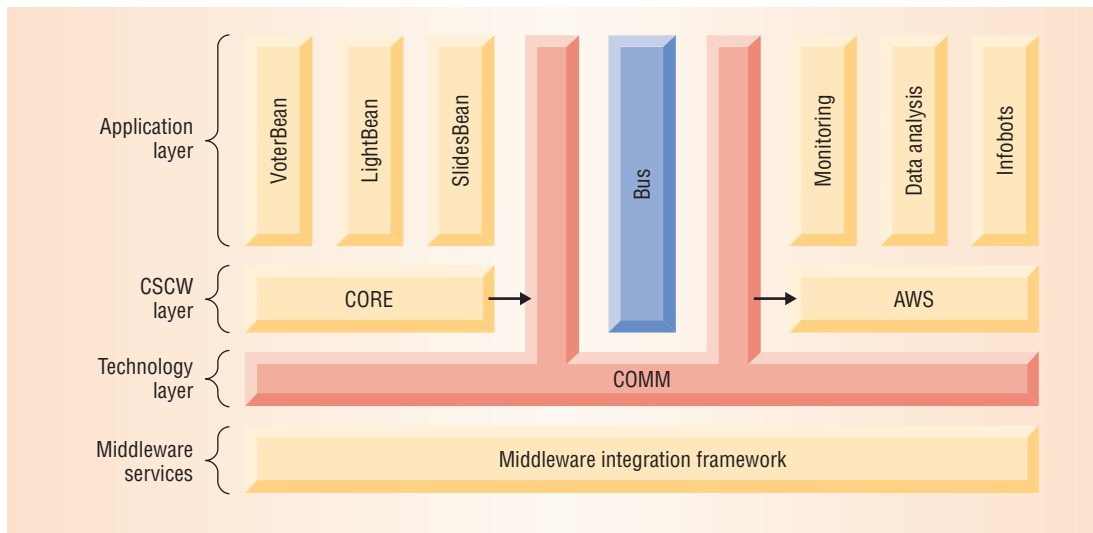


Figure 1. ANTS architecture. All components trigger events to the collaboration bus, and awareness components glean system changes from it.

ponent model that integrates with a flexible monitoring service. Further, the developers have not constructed their models atop a middleware integration framework, so broadcasting and persistence services have to be created ad hoc.

Another important trend in groupware toolkits underscored the importance of a *collaboration bus* as an essential service either for supporting state propagation and event management or for monitoring awareness systems. For example, Prasun Dewan's Collaboration Bus project at the University of North Carolina at Chapel Hill (www.cs.unc.edu/~dewan/cb.html) developed a software abstraction that facilitates composing collaborative systems by providing generic services such as session and coordination support and shared data structures.

ANTS FRAMEWORK

To address the deficiencies with existing toolkits, we created the ANTS CSCW component framework (<http://ants.etse.urv.es/>). We based ANTS on refined versions of all three groupware development approaches, proposing new solutions at the conceptual, architectural, and technological levels.

The distributed Model View Controller in Bellcore's Rendezvous project⁶ also influenced our work, as did monitoring and awareness systems such as Nessie.⁷ Designed to provide a generic multiuser collaborative framework, ANTS is based on solid distributed technologies and provides generic CSCW services to the user and developer communities.

The framework defines a dedicated awareness

and monitoring service that seamlessly integrates with the overall component model. Based on the standard JavaBeans specification, the component model hides complexity from users by providing transparently remote persistence, distributed events, and component descriptors and packaging. The monitoring services and component model are positioned atop a middleware integration framework, which uses a standardized approach to distributed services. By combining centralized and replicated architectural services, ANTS provides developers with a variety of tools and architectural designs to manage different problem sets.

The overall architecture consists of three main layers: a technology layer, a CSCW layer, and an application layer. Upper layers represent high-level abstractions that hide the complexity of the layers below. As Figure 1 shows, all components trigger events to the collaboration bus, and awareness components listen to it for information about what is happening in the system.

Application layer

Our primary goal is to facilitate the development of collaborative components and applications or monitoring agents by achieving a smooth transition from local to distributed applications. The application layer abstracts access to a remote property-based persistent component and distributed events produced in the current session. Like many other groupware toolkit developers, we use Java and JavaBean as our preferred technologies. Our approach sustains JavaBean components in a con-

CSCW Services

In creating the ANTS framework, we focused on four essential CSCW services: shared sessions, support for synchronous and asynchronous components, coordination, and awareness.

A *session* is a group of objects associated with some common communications pattern, supporting full-duplex multipoint communication among an arbitrary number of connected application entities. The concept of a shared session is essential to any CSCW toolkit, and it defines the basis for shared interactions in a common remote context.

Explicit support for *shared artifacts* is also essential. Normally, users engage in collaborative interaction with other users in a shared context, or with accessible collaborative components. These collaborative components—usually called artifacts—range from synchronous to asynchronous, depending on the interaction time. Explicit support for the creation and insertion of new artifacts is a differentiating factor between extensible and nonextensible CSCW toolkits.

Another key issue is explicit software support for *coordination policies*. These policies can be set programmatically or declaratively, and they can be categorized in rules for access, concurrency, and floor control. The platform should provide extension hooks to insert custom coordination components that would then transparently interpose in normal calls to a collaborative component.

Finally, collaborative environments require *awareness*, which can be defined as “an understanding of the activities of others, which provides a context of your own activity.” From our perspective, an awareness platform should facilitate data acquisition from the running environment.

tainer that hides complexity in distributed programming and networking skills.

The ANTS server stores properties as shared data structures. Each JavaBean component can use string properties, string indexed properties, and serializable object properties. To assure component persistence, the remote property-based component stores these properties in a database. Each property change will trigger an event that allows a later subscription to `PropertyChangeEvent`s. In addition, `addPropertyChangeListener` methods receive property-change events in bound properties. These events are received from the distributed notification service and restricted to events in the current session and component.

The framework supports coordination generically by providing coordination rules for access, concurrency, and floor control. Each method can trigger a `PropertyVetoException` similar to the JavaBean model’s constrained-property convention. Our model differs, however, in the source of the `VetoException`: In ANTS, registered coordination managers interpose calls to a component to assure the requested action’s validity.

An XML descriptor file facilitates ANTS framework customization and component introspection. In general, a descriptor includes properties and event information. Event information provides the awareness infrastructure with the key for knowing which events are triggered by each resource in the shared environment.

Component packaging uses standard jar files similar to the JavaBean component model. The package must contain the XML descriptor, component classes, and any local resource, such as images or files, used in the component. An automated process uses the XML descriptor to register components in the component repository, deploys the packaged components to the ANTS server, and copies unpackaged resources to the Web container. The repository controls the permissions for getting information about registered components and allows insertion and removal of artifacts in shared-session contexts.

CSCW layer

The “CSCW Services” sidebar describes the set of collaborative services that the ANTS container offers to the application layer. Two main modules seamlessly interconnect via the collaboration bus. CORE is a runtime container with a generic and extensible application layer for collaborative tools, while AWS is a generic awareness service that provides appropriate actuators for events received from the collaboration bus.

CORE. This module includes explicit support for sessions, shared artifacts, coordination control, security, and integration for both synchronous and asynchronous applications. Its design was strongly influenced by their extensible and generic multi-user object-oriented (MOO) architectures. In a MOO system, *place* objects, which represent discrete locations interconnected by portal objects, define the space’s topology. These portal objects can establish graph-directed relationships between places. Every static or dynamic entity is an object or *thing*, which has a set of properties and a set of verbs that can be added or removed at runtime.

As in a MOO system, the place object represents an ANTS shared session, interconnected by portal objects. JavaBean components, dynamically loaded by the ANTS container, represent the shared artifacts. Object properties are represented by the remote, persistent property-based component, and object verbs are JavaBean classes.

The place object represents a shared session and contains users, components, and links to other places. It provides methods to send or subscribe to

events in the current context. The place object also supplies methods to get connected users and available links to other places. In addition, it permits dynamic loading of JavaBean components to the shared context.

The JavaBean component model provided inspiration for defining our coordination managers. In our approach, these special listener objects can approve or veto property changes on constrained properties. At this stage, we provide sample access-control and floor-control coordinators. We also supply a centralized token-based lock component that provides the base for many floor-control coordinators.

AWS. Based on existing CSCW awareness systems,⁷ the server-side awareness infrastructure enables triggering of a set of *actuators* that perform special tasks in response to sensors. Sensors represent any physical or software component that produces and transmits events to the notification system. The *mediator* binds a sensor with one or more actuators and launches actuators in response to events.

Event sensors activate in response to application events that the information bus produces. AWS creates an event sensor by using a subscription that follows the constraint filter language available in the notification system. The time scheduler activates *time sensors*, which can be created for a specific date or even set to trigger on repetitive dates. The time scheduler notifies the awareness service when it activates a time sensor.

XML descriptors included in packaged components seamlessly integrate AWS with the component model. Every descriptor contains the events that the component triggers to the bus. Because the repository stores this information during the deployment phase, external monitoring applications or AWS actuators can be aware of the information that every component in the platform produces.

AWS provides the basis for sophisticated monitoring applications, intelligent agents, and persistent actuators activated by events received from the notification system. For example, we could create an event sensor subscription that says we are interested in events produced by a user named Pedro. We could then make the sensor persistent with an actuator that would log each event Pedro initiated in the environment. We could also create our own bots, which would react to events that our subscription specifies.

Technology layer

Given that many CSCW systems rely on nondistributed, single-threaded environments that limit their

applicability to more complex problems, we chose the Java 2 Enterprise Edition standard as our technology infrastructure. Using J2EE avoids the need to implement infrastructure- and system-specific code and lets us base our work on open specifications and components. Our system is thus vendor-independent: Developers can choose any J2EE-compliant application server or relational database.

Our framework's message-oriented middleware also needed a publish-subscribe notification service. Java Message Service (JMS) offered the standard alternative, but the Elvin notification service also offers good performance and has been used for CSCW purposes. We therefore implemented a façade API, COMM, that lets us choose between any JMS-compliant messaging solution and Elvin.

We provide several sample bus aliases for developers to connect to different middleware services. Depending on the specific requirements and the selected middleware, we can choose different transport layers such as raw multicast, reliable multicast, encrypted Secure Socket Layer streams, or ordered streams. Nonexpert programmers can simply select SSL as the bus alias and thus use an encrypted communication channel. Applications requiring scalability for many users could then select the RMCast alias.

IMPLEMENTATION

To prove our concept's viability, we have implemented several projects using ANTS, including a shared whiteboard, multiuser games such as *Pong*, and a MOO text interface. The Multiuser-Oriented Virtual Environment (<http://ants.etse.urv.es/move>) and Innovative Technologies for Collaborative Learning and Knowledge Building (<http://www.euro-cscl.org/site/itcole>) highlight the benefits of the key design decisions that shaped the framework. These benefits include a component-based approach that simplifies development of collaborative artifacts and applications, a monitoring service for creating event-analysis tools and agents that react to information events in the collaboration bus, and a system that runs—transparently to users—atop advanced distributed services that assure scalability and performance under high loads.

Move!

Part of the Catalan Internet2 project (www.i2cat.net), Move! is a three-dimensional collaborative virtual environment that lets user avatars interact with one another or with shared artifacts

AWS provides the basis for sophisticated monitoring applications, intelligent agents, and persistent actuators.



Figure 2. Move! components. The relatively undemanding 2D and 3D presenters, voting tool, and bars tool (top) use a centralized property store. The computationally demanding avatar control (bottom) uses a replicated approach that directly accesses the collaboration bus for state propagation.

such as presenters or 3D simulations. As an educational tool for use in medicine, archaeology, and other disciplines, Move! supports many users and permits visualization and interaction with computation-intensive 3D simulations. Built on open technologies such as the Virtual Reality Modeling Language and the H-Anim humanoid specification, Move! demonstrates how well ANTS handles complexity and software architecture.

Application level. Every shared artifact in the

Move! environment is implemented as a framework component that must be packaged and deployed for later use. As Figure 2 shows, depending on their specific performance requirements, components can follow either a centralized or replicated approach. The ANTS distributed model-view-controller pattern facilitates developing different views for the same component. Move! includes both 3D and text-based views of the shared scenarios.

CSCW level. Framework components represent the shared artifacts, and coordination through tokens assures consistency in multiuser access to a shared object. As Figure 3 shows, AWS forms the basis for the Move! agent infrastructure, while the bus provides state-propagation and awareness functions. Developers can program bots that perform special tasks and react to events in the environment. For example, when a user moves beside a guide bot, it shows the scene and interacts with the relevant artifacts. We are using more advanced agents pervasively to test the environment under high loads.

Technological level. Our neutral approach to middleware lets developers choose from among a wide range of application servers, database engines, and notification services. In this case, we chose the Oracle database, Orion application server, and Elvin notification service, but we could easily replace any of these with competing products. Move! also illustrates the benefits of advanced middleware features such as application server clustering, load balancing, and Elvin event performance. By moving the processing burden to the underlying middleware, we can focus on implementing the CSCW framework services.

Performance issues. Tests using Move! reveal that the critical point as far as performance resides on the client side, not in the ANTS platform or server-side middleware. Each user connected to the system requires small memory and processing-power allocations. Further, the notification system can handle massive event flows without degradation, thus achieving sufficient scalability under high loads. Although Move! scales up smoothly with 200 users in a shared session, the Internet2 project-usage scenario provides better hardware and bandwidth resources that can sustain more users.

On the client side, the 3D visualization engine requires large amounts of memory when it handles complex simulations or avatars. Moreover, the arrival of many events at the client creates problems with transmitting to the visualization engine, which can lead to system degradation. To solve this problem, we implemented a distance-based algo-

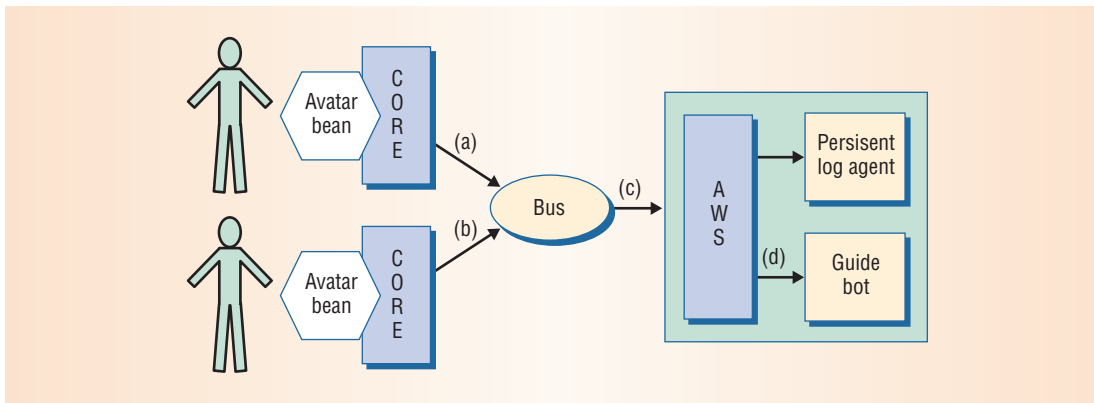


Figure 3. Move! services. (a) Move! sends the user's movements to the bus, which (b) transmits them to the other clients in the shared context. The mediator monitoring service (c) listens to the bus and launches specified actuators in response to filtered events—in this case, (d) the mediator activates a guide bot that begins a chat interaction with the specified user.

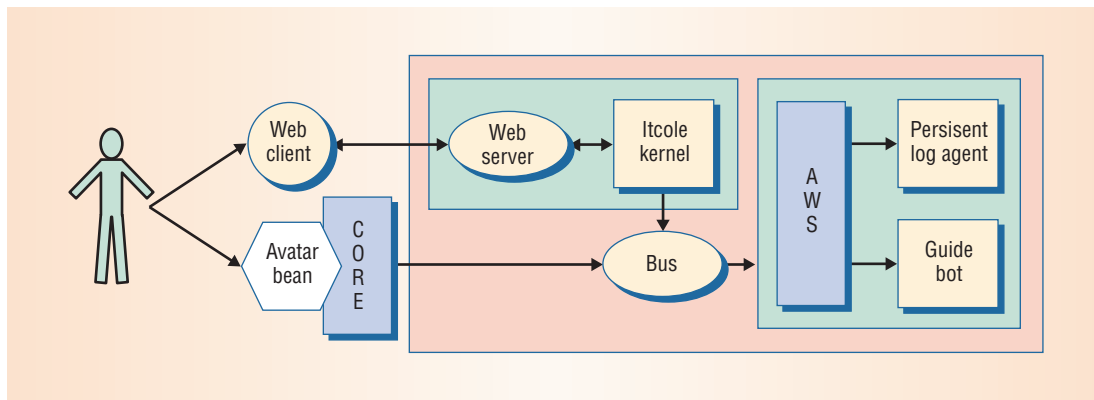


Figure 4. Itcole architecture. This ongoing project provides a seamless infrastructure for synchronous applications and monitoring agents to create a collaborative learning environment.

rithm that discards events located outside a specified radius around each user. This strongly limits unnecessary processing on the client side and lets the system handle many concurrent users.

Itcole

Another ongoing project, Itcole applies ANTS services and concepts to create a seamless infrastructure for synchronous applications and monitoring agents in a collaborative learning environment. Itcole includes a knowledge construction tool, synchronous components such as a shared whiteboard, and event analysis applications. The user can interact asynchronously with the environment through a Web client or collaborate synchronously through Java applets.

Figure 4 shows the Itcole architecture. Both synchronous and asynchronous applications send

events to the collaboration bus, enabling monitoring applications to listen to a unique information stream and react in specific ways. Synchronous components such as the map applet use the bus to propagate state changes. Monitoring applications, such as persistent log agents or tutor bots, can be developed atop AWS to react to information produced in the bus, providing a flexible scenario for studying the running environment.

We foresee research in the application of machine learning techniques across areas such as clustering and the development of event analysis tools to study information flows in the collaboration bus. The confluence of both synchronous and asynchronous events can help researchers better understand the collaboration process. This aspect plays an even more important role in the Itcole project, which requires analyzing collaborative learning scenarios.

ANTS constitutes an ambitious application framework that simplifies development of distributed collaborative components in the Java language. A major drawback of the current platform, however, is its Java-centered approach. Therefore, we are using XmlRpc and XML streams to create an XML system interface that supports other user interfaces and programming languages. Further, future support for Web services in application servers could facilitate system interoperability.

The generic concepts we have described can easily be mapped to other non-Java component models. Consequently, we could replace JavaBeans with other client-side component models, using a non-Java middleware integration framework to replace the J2EE application server. Ultimately, we foresee challenging research work in advanced middleware services—constructed atop overlay networks—as the basis for the next generation of collaborative systems. ■

Acknowledgments

This project was partially funded by the European Project ITCOLE IST-2000-26249 and the

Generalitat de Catalunya Internet2 Advanced Telecommunications project.

References

1. P. Dourish, *Open Implementation and Flexibility in CSCW Toolkits*, doctoral dissertation, Dept. Computer Science, University College London, London, UK, 1996.
2. M. Roseman and S. Greenberg, "Building Real-Time Groupware with GroupKit, a Groupware Toolkit," *ACM Trans. Computer-Human Interaction*, vol. 3, no. 1, 1996, pp. 66-106.
3. A. Chabert et al., "Java Object-Sharing in Habanero," *Comm. ACM*, vol. 41, no. 6, 1998, pp. 69-76.
4. H. ter Hofte, *Working Apart Together: Foundations for Component Groupware*, Telematica Instituut Fundamental Research Series, no. 001, Telematica Instituut, Enschede, the Netherlands, 1998, p. 288.
5. I. Marsic, "DISCIPLINE: A Framework for Multimodal Collaboration in Heterogeneous Environments," *ACM Comp. Surveys*, vol. 31, no. 2es, 1999; <http://portal.acm.org/citation.cfm?doid=323216.323225>.
6. R.D. Hill et al., "The Rendezvous Architecture and Language for Constructing Multiuser Applications," *ACM Trans. Computer-Human Interaction*, vol. 1, no. 2, 1994, pp. 81-125.
7. W. Prinz, "NESSIE: An Awareness Environment for Cooperative Settings," *Proc. 6th European Conf. Computer Supported Cooperative Work (ECSCW 99)*, Kluwer Academic, 1999, pp. 391-410.

Pedro García López is an assistant professor in the Department of Computer Engineering and Mathematics at the University of Rovira i Virgili, Spain. His research focuses on distributed systems and middleware infrastructures for computer-supported collaborative work and learning. García received a PhD in computer engineering from the University of Murcia, Spain. He is a member of the ACM. Contact him at pgarcia@etse.urv.es.

Antonio F. Gómez Skarmeta is a professor in the Department of Computer Science, Artificial Intelligence, and Electronics at the University of Murcia. His research interests include fuzzy modeling, distributed artificial intelligence, telelearning and computer-supported collaborative work, and telematics services in broadband networks. Skarmeta received a PhD in computer science from the University of Murcia. He is a member of the IEEE Computer Society. Contact him at skarmeta@fcu.um.es.