Leveraging the Linda Coordination Model for a Groupware Architecture Implementation

José Luis Garrido[1], Manuel Noguera[1], Miguel González[2], Miguel Gea[1], and María V. Hurtado[1]

[1] Dpt. Lenguajes y Sistemas Informáticos, University of Granada, E.T.S.I.I., c/ Saucedo Aranda s/n, 18071 Granada, Spain
{jgarrido, mnoguera, mgea, mhurtado}@ugr.es
[2] Tecnologías de la Información, Autonomous University of Madrid, E.P.S., c/ Tomás y Valiente 11, 28049 Madrid, Spain
miguel.gonzalez@uam.es

Abstract. Functional and non-functional requirements must be taken into account early in the development process of groupware applications in order to make appropriate design decisions, e.g. spatial distribution of group members and group awareness, which are related to the main characteristics exhibited by CSCW systems (communication, coordination and collaboration). This research work presents a proposal intended to facilitate the development of groupware applications considering non-functional requirements such as reusability, scalability, etc. In order to achieve these objectives, the proposal focuses on the architectural design and its implementation, with emphasis on the use of a realization of the technological Linda coordination model as the basis for this implementation. The outcome is a distributed architecture where application components are replicated and event control is separated. This work is part of a conceptual and methodological framework (AMENITIES) specially devised to study and develop these systems.

1 Introduction

Groupware has been defined as "a computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment" [7]. To date, groupware has comprised various systems: Workflow Management Systems (WfMS), Computer-Mediated Communication (CMC) (e.g. e-mail), Decision Support Systems (DSS), shared artifacts and applications (e.g. shared whiteboards and collaborative writing systems), etc. These systems include common and specific requirements in relation to the following group activities [7]:

- *Communication*. This emphasizes the exchange of information between remote agents by using available media (text, graphics, voice, etc.).
- *Collaboration*. Effective collaboration requires people to share information in the group context.
- *Coordination*. The effectiveness of communication/collaboration is based on coordination. It is related to the integration and harmonious adjustment of the individual work effort towards the accomplishment of a greater goal.

The inherent complexity of CSCW (Computer Supported Cooperative Work) systems requires a great deal of effort in specifications and development [3]. The development of groupware applications, the technological part supporting collaboration processes in CSCW systems, is more difficult than that of a single-user application; social protocols and group activities must be taken into account for a successful design [13]. Methodologies and implementation techniques aimed at enhancing group interaction activities (especially for synchronous groupware [23]) should therefore be applied. On the other hand, there is a lack of methodological proposals for addressing and integrating the study and development phases of a CSCW system. Furthermore, we argue that special attention must be paid to the software architecture, which is defined by the recommended practice for architectural description of software systems [2] as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution".

This article shows the implementation of a high-level architectural design for groupware applications [11]. This architecture guarantees important software quality properties since the main design criterion centers on mutual component independence. To this aim, we propose the use of the data-driven programming model [20] provided by the JavaSpaces technology [29], a realization of the Linda coordination language [5], in order to accomplish its implementation. The proposal based on this technological data-driven coordination has two aims: firstly, to fulfill common and specific functional requirements in CSCW systems related with human communication, coordination and collaboration, such as to provide group awareness in order to support and enhance these group activities; and secondly, to be able to build this kind of distributed applications taking also into account non-functional requirements such as reusability, scalability, etc. This proposal is part of AMENITIES [10], a conceptual and methodological framework that is specially devised to study CSCW systems and develop groupware applications.

The paper is organized as follows. Section 2 briefly introduces the AMENITIES methodology, providing a general description of its models and stages. Section 3 focuses on how an architectural design enables us to address the development of groupware applications. Section 4 introduces the Linda coordination model and describes how it is used to implement group awareness in this architectural design. In Section 5, the physical architecture and its corresponding deployment are described briefly for the current implementation. Section 6 references related work and the main conclusions are provided in Section 7.

2 AMENITIES

AMENITIES [10] (an acronym for A MEthodology for aNalysis and desIgn of cooperaTIve systEmS) is a methodology based on behavior and task models, specially devised for the analysis, design and development of CSCW systems. The methodology stems from cognitive frameworks (activity theory, distributed cognition, etc.) and methodological proposals (requirements [17] and software engineering [27], task analysis [31] and modeling [21]). Thereby, AMENITIES provides a conceptual and methodological framework that seeks to avoid the main deficiencies found in approaches traditionally applied to this kind of system, by focusing on the group concept and covering the most relevant aspects of its behavior (dynamics, evolution, etc.) and structure (organization, laws, etc.). Another objective is to allow us to systematically address the analysis and design of CSCW systems and to facilitate subsequent software development. Therefore, it also proposes a concrete methodology including concrete phases, models, notations, etc. Fig. 1 provides a general scheme of the methodology, showing the main models (boxes) and stages (dashed lines) involved. Just like most methodologies, AMENITIES follows a simple iterative process allowing us to refine and review these models.

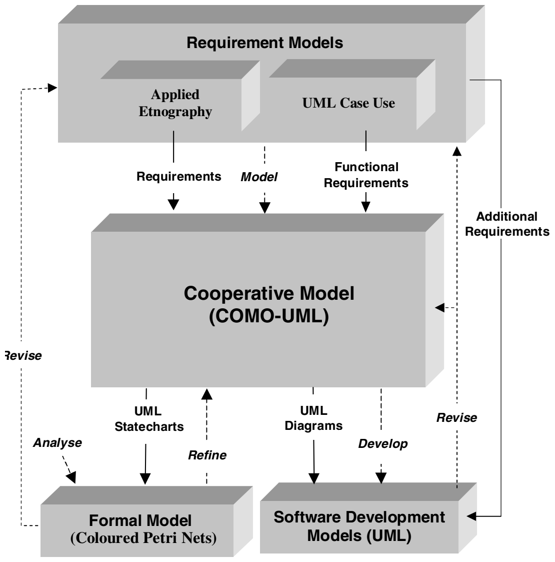


Fig. 1. General scheme of AMENITIES

The cooperative model (called COMO-UML in Fig. 1) is the core of the methodology and enables us to represent and connect instances of concepts defined in the conceptual framework of AMENITIES, according to the requirements for each specific system. The cooperative model [8] describes the system (especially on the basis of coordination, collaboration and communication) irrespective of its implementation. It therefore provides a better understanding of the problem domain. In order to build this model, a structured method (comprising four stages) is proposed: specification of the organization, role definition, task definition, and specification of interaction protocols. This method has been specifically devised to make easier connections between all the concepts (for instance, tasks to be performed under each role). The modeling notation proposed is based on both UML state and activity diagrams [19], but with a semantics specially defined for this problem domain [9].

3 An Architecture for the Groupware Development

The architectural design of the groupware application is the starting model for the software development phase in AMENITIES, and therefore, for the proposal introduced in this paper. This section shall first present the motivations and foundations for our architectural proposal; it shall then make the proposal concrete by using a real case study.

3.1 Motivations and Foundations

Most requirements to be considered for the development of each specific groupware application are specified in the cooperative model described above, for instance, tasks requiring various actors to be accomplished (i.e. cooperative tasks), or constraints (specified by means of the law concept) preventing an actor from being involved in more than one task. However, apart from specific requirements for each groupware application, common design issues for this kind of system should be taken into account at an abstract level. An architecture guiding the organization of architectural elements (basically composition, interfaces and interactions) can just provide this desired abstraction level covering the following general objectives:

- Groupware applications are inherently distributed. It is therefore important to obtain an implementation stemming from a set of subsystems that communicate with each other through well-defined interfaces. The division/partitioning of the whole system into components (called subsystems) facilitates its development, evolution and maintenance.
- Appropriate organization and mapping of functionality onto subsystems in order to achieve certain desired software properties such as reusability, portability and interoperability.
- A groupware system should be able to increase the number of subsystems because new applications supporting other activities could be added for the same group.

In order to achieve these objectives, some guidelines of the Unified Software Development Process [15] for specifying the architectural view of the design model have been adopted. For this purpose, the design process is carried out using the UML language, providing the three following architectural views:

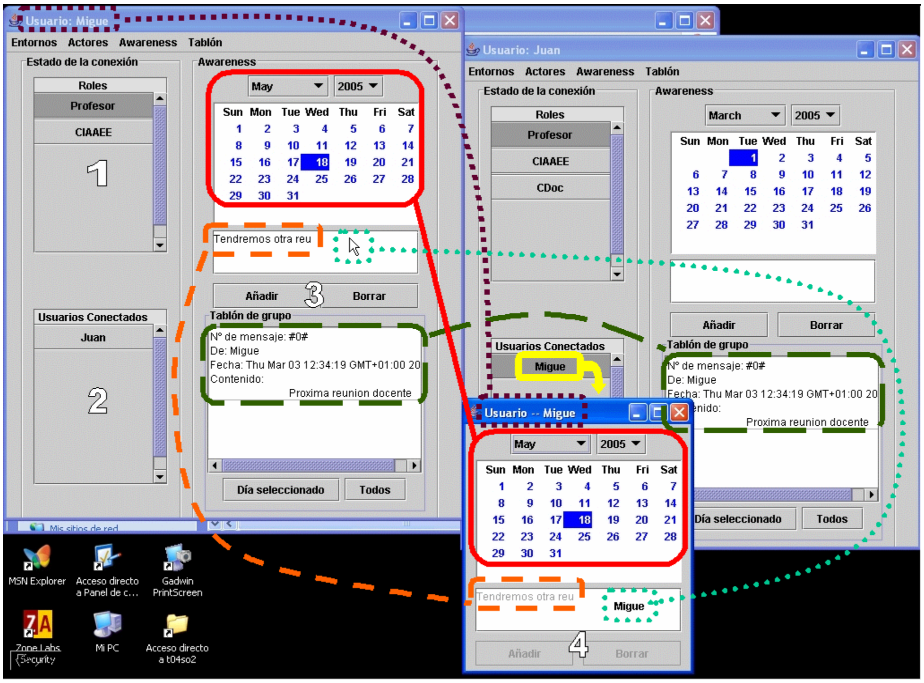
1. *Component view*. In order to represent the system partitioning, package diagrams with the stereotypes system and subsystem are used in conjunction with the composition relationship (a form of the aggregation association that considers bound parts by lifetime).
2. *Functional view*. The functional aspects (static structure) of the system are specified by means of both class and interface diagrams associated with subsystems, and creating connections between them on the basis of the use relationship.
3. *Behavioral view*. System behavior (dynamic view) (i.e. how the subsystems collaborate by means of interactions) is specified on the basis of the functional structure (described in the previous paragraph), using collaboration diagrams.

3.2 Case Study

A collaborative appointment book application for group work has been developed according to the architecture to be described in the next subsection. This application allows lecturers/researchers (people playing these organization roles) within the same department at the University of Granada to coordinate in proposing meetings in a common forum. This human coordination must be supported on the basis of providing group awareness [25]; users currently connected to the system can observe each other (both presence and activity). An additional requirement for the system is to allow participants to share information in real-time (synchronous) and asynchronous modes.

The application (see Fig.2) consists of:

1. A panel for possible roles to be played, which also highlights the role currently being played by the participant.
2. A panel showing the other participants who are playing the same role.
3. A panel including the calendar and the messages sent.
4. A popup window to show the activities of each participant we are interested in.



Nº de mensaje: #0#
De: Migue
Fecha: Thu Mar 03 12:34:19 GMT+01:00 20
Contenido:
Proxima reunion docente

Fig. 2. General view of the collaborative appointment book application

3.3 Architecture

According to the objectives and the three views described in the subsection 3.1, the architecture is made concrete as follows:

1. *Component view*. The whole system is divided into several parts. A basic groupware application consists of four subsystems: *Identification*, *Metainformation*, *Awareness* and the application itself (in this case, the *Appointment book*). Fig. 3 shows the component diagram for the case study, illustrating the UML package diagram and the *System* and *Subsystem* stereotypes. Although there is only one application in this example, other applications can be easily integrated while this design is maintained.

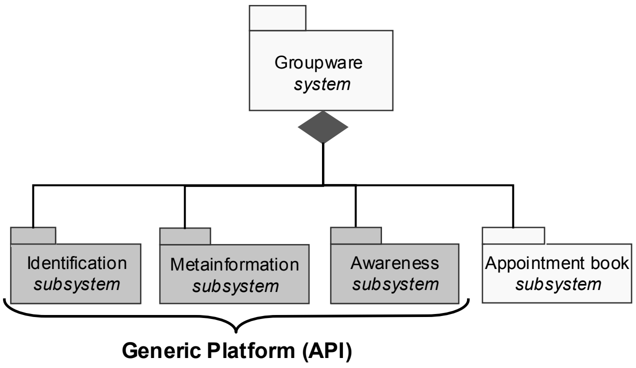


Fig. 3. UML package diagram for the component view

The *Identification*, *Metainformation* and *Awareness* subsystems are always present for every groupware application. The application subsystem is obviously specific for each groupware application. All the subsystems are described in more detail in the next views, according to the functionality they provide and the behavior exhibited.

2. *Functional view*. By means of a UML interface diagram, Fig. 4 (left) shows the four mentioned subsystems and their use relations on the basis of the associated functionality. Each subsystem provides an interface designed to achieve independence between the subsystems and other applications making use of them.

   In particular, the *Metainformation* subsystem supports all the functionality for checking metadata (possible roles to be acquired, laws applied by the organization, etc.) specified in the cooperative model. The *Identification* subsystem is used to start the application and to control users' access to the system. The *Awareness* subsystem is intended to maintain shared and contextual information (telepointers, list of participants playing a specific role, etc.) in charge of providing group awareness that the participants need for an effective collaboration. Finally, the *Appointment book* subsystem provides both an extended functionality as that of a single-user appointment book and slightly different semantics.

3. *Behavioral view*. Fig. 4 (right) shows how the subsystems collaborate to resolve interactions between users and the *Appointment book* subsystem. A UML collaboration diagram is used to model this behavior.

   In this case, a typical user interaction starts at the *Appointment book* subsystem. Firstly, users must identify themselves in the system (messages 1, 1.1, and 1.2). Once they have been correctly identified, they can choose to register their presence in the system so that other users can choose to observe them (messages 2, 2.1, and 2.2). They themselves want to know which users are currently connected under the same role (messages 3, 3.1, 3.2, and 3.3). The system therefore supplies the necessary infrastructure for group awareness and we are able to collaborate in real time.

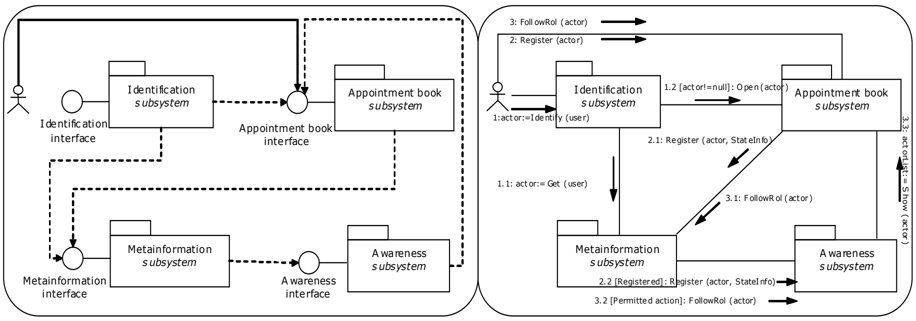

Identification subsystem
Identification interface
Appointment book subsystem
Appointment book interface
Metainformation subsystem
Metainformation interface
Awareness subsystem
Awareness interface
3: FollowRol (actor)
2: Register (actor)
1:actor:=Identify (user)
1.2 [actor!=null]: Open(actor)
1.1: actor:= Get (user)
2.1: Register (actor, StateInfo)
3.1: FollowRol (actor)
3.3: actorList:= Show (actor)
2.2 [Registered]: Register (actor, StateInfo)
3.2 [Permitted action]: FollowRol (actor)

Fig. 4. UML interface diagram (left) and collaboration diagram (right)

4 Architecture Implementation

There are basically three ways of communicating/sharing information between group members in order to provide effective group awareness: "*explicit communication, where people tell each other about their activities; consequential communication, in which watching another person work provides information as to their activities and plans; and feedthrough, where observation of changes to project artifacts indicates who has been doing what*" [14].

In the following subsections, we will briefly describe Linda and JavaSpaces (Section 4.1), and also how the Linda coordination model is applied to the implementation of the *Awareness* subsystem (Sections 4.2 and 4.3) within the architectural design presented in Section 3.3. The focus is on how object-oriented tuple space features can be used to support these three ways of communicating/ sharing information.

4.1 The Linda Coordination Model and the JavaSpaces Technology

The aim of distributed computing is to design and develop each distributed application as a set of processes and data which are distributed over a computer network, and to interact with them in an integral way. Computation and involved element coordination are usually addressed separately in distributed computing. A technological coordination model establishes the relations between components and in turn provides the mechanisms needed to enable interaction between them. These mechanisms are orthogonal to the computation model [12].

The space-based distributed computing model is derived from Linda [5]. A space is an object (or tuple in Linda terms) store for data shared through a computer network. A tuple is a data structure with several typed fields set to particular values, e.g.

<10,"Madrid">, consisting of an integer number and a string. Only a few atomic, basic functions are provided to operate on them (see Table 1). Operations *in* and *rd* use an especial template tuple as a pattern with which to match those tuples to be retrieved. If several tuples match the searching template, only one is retrieved non-deterministically. Since a tuple space is "global" (i.e. visible from any location), the processes on any computer can insert tuples into or take tuples from the space concurrently.

Table 1. Tuple space operations

Operation	Description
out	Insert a tuple into the space
in	Take a tuple from the space
rd	Inspect (read) a space tuple, without removing it

JavaSpaces [29] has been chosen as a tuple space implementation based on Linda. In JavaSpaces, tuples and their fields are typed as objects in the Java programming language. This provides a richer type system than Linda does. Since tuples are objects which belong to a particular class, they may have methods which are associated with them.

There are four main operations which can be invoked in a JavaSpace tuple space: *write, read, take* (can be blocking or not) and *notify*; the first three operations correspond to the Linda operations *out, rd,* and *in*. Henceforth, we will use the operation names provided by JavaSpaces. Objects use the *notify* invocation to ask the space to inform them that one kind of tuple matching a given template has been inserted into the space. The mechanism involved in this operation is called distributed notification of events [28]. When an object wants to be notified about some kind of tuple insertion, it must register the corresponding matching pattern of the tuple together with the *notify* operation. Notification will be provided when a tuple matching this pattern has been inserted into the space. This mechanism will be very useful as basis of providing context awareness in groupware applications.

In JavaSpaces, the operations mentioned above are not limited to a single space but may be carried out in turn on different spaces. As far as the information space (i.e. the tuple space) is concerned, this also enables centralized, replicated, dynamic, hybrid, etc. architectures to be devised. On writing a tuple in the space, the time it will remain in the space before being deleted is also defined. This time may be indefinite so the tuple in question will remain until it has been withdrawn by the *take* operation.

4.2 Data Sharing

While the cooperative task is being performed, it is necessary to share data. This section discusses the basic mechanism to share data stored in tuples. Depending on when data are shared, it is possible to distinguish between synchronous and asynchronous modes of communicating information [23]. Synchronous communication shows the changes that occur in the shared environment as soon as they take place. Asynchronous communication can be obtained for instance, when logging into the system and observing the changes that other users have made on the shared objects at another time.

In the collaborative appointment book application, both types of communication can be found in the way the user interacts with the board saving appointment messages. When a user decides to log into the system and download the corresponding board messages, he/she can see the contributions that other group members have submitted previously (asynchronous communication). If he/she wants to be informed of any message arriving at the shared board, he/she will immediately see any contribution from the members playing their current role (synchronous communication). Similarly, users can choose what kind of system interaction they desire at any time and effortlessly change from one to another.

Fig. 5 shows a sample scenario for the collaborative appointment book introduced in Section 3.2. Let us imagine that there are several users in the system (e.g. *Anna*, *Peter*, *John* and *Martha*). Each one is part of at least one group. For example, all are members of the group *Prof* (*Professor*) and *Peter*, *John* and *Martha* are also *TC* (*Teaching Committee*) members. Let us imagine that user *John* has just logged into the system. At present, only *Anna*, *Peter* and now *John* are connected. If *John* decides to load his group board, he will see all the proposals that other group members have previously submitted. In this example, we can see how he would read *Peter's* and *Martha's* messages (*Martha* is not even connected now). If he decides to "register" to be notified about the submission of new messages by other members, he will need to insert a template tuple into the space (through his *Awareness* subsystem) in order to show that he is interested in the tuples related to his group board.

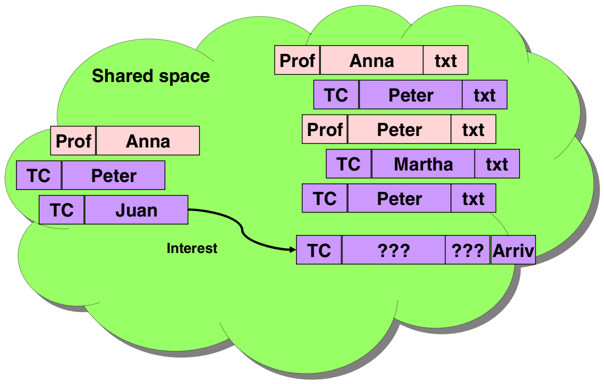


Fig. 5. Synchronous and asynchronous data communication

4.3 Feedthrough and Consequential Communication

The JavaSpaces mechanism for distributed notification of events (*notify* operation described above) and the definition of the time a tuple may remain in the space are also used in our approach to support the other ways of group awareness, namely feedthrough and consequential communication. This section discusses how events are propagated in the system.

We have found that a groupware system can be basically modeled using two types of tuples: tuples containing steady information and which are inserted into the space for an indefinite time; and tuples used to announce events and which are more volatile since information relating to their associated events is relevant at a given time and in a particular context. This is shown through the two following scenarios.

Actor's Presence

Let us consider that the actor *John* has logged into the system and is playing the role of *Teaching Committee* (*TC*) member. He is reading this committee board to see what messages have been sent by other members. *Peter* is currently connected and he is playing the same role as *John* (i.e. *TC*). *Anna* is also connected and playing the role *Prof*. In addition, *John* would like to know which other members are connected now in order to discuss the date of an online meeting with them or to see which other proposals they are currently submitting. In order to accomplish this, user *John* asks (from his appointment book application, and therefore, the appointment book subsystem) to be informed of which other users playing the same role as him are connected. In turn, the *Appointment book* subsystem will use the awareness subsystem to insert appropriate template tuples into the space so that he will be notified whenever any user playing the role *TC* arrives or leaves. This scenario is depicted in Fig. 6. The same applies to the notice board messages that are represented on the right-hand side of the figure.

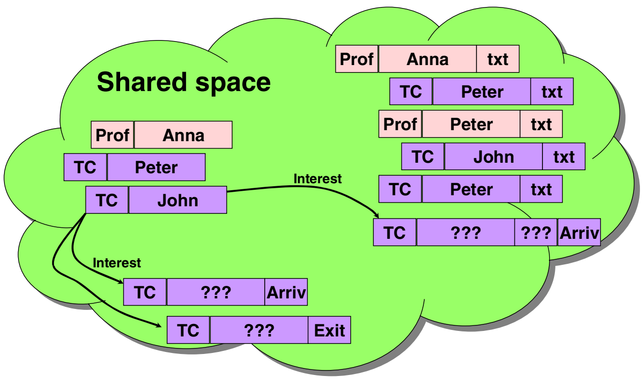


Fig. 6. Starting scenario

If professor *Martha*, who is also a *TC* member, enters the system playing this role, the space would notify *John's Awareness* subsystem of this event. Previously, *Martha's Awareness* subsystem would have inserted the appropriate tuples into the space (see Fig. 7) when she logs in.

In this way, *Martha's Awareness* subsystem would insert a (1)-type tuple (as mentioned at the beginning of this Section) indicating more stable information such as her name, role she is playing, connection time, etc. (for the sake of simplicity, this tuple has the form <TC, Martha> in Fig. 7); and another (2)-type tuple in the form <TC, Martha, Arriv> that provides "event" information such as "Martha has just logged in". The (2)-type tuples would have a brief remaining time in the space associated since this kind of information is only meaningful or valid to the connected users that have asked to be notified of other group member activity. If after half an hour, another *TC* member connects (e.g. *Martin*), his *Awareness* subsystem would inform the other members playing his role by reading <TC,???> tuples.

The same applies to the exit notification. When *Martin* logs in, the tuples used by other members logging out before he connects (e.g. <TC,Peter,Exit>) are useless and must therefore be deleted. By deleting these tuples, we also decrease the information overload in the space. We will see this in detail in the following scenario.

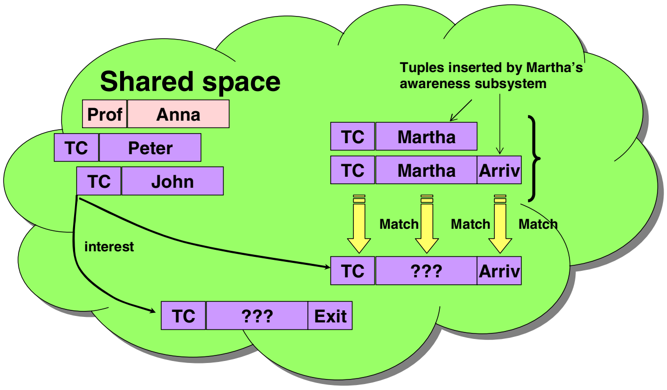


Fig. 7. Arrival of a member with the same role

Actor's Activity

The need for tuple elimination is more evident if we consider the telepointer mechanism. The coordinates of other users' mouse pointers have a very brief validity (a hundredth of a second at most) since they are continuously changing. In this regard, the programmed removal of the tuples in the space is a very efficient way of maintaining space consistency and preventing obsolete information being stored.

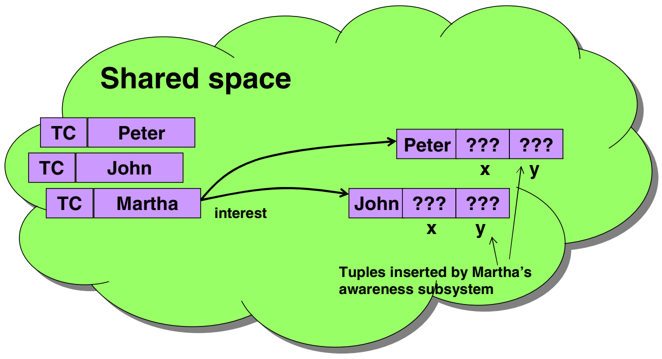


Fig. 8. Remote mouse pointer tracing

The starting point for the following scenario is Scenario 1, although some tuples have been hidden for readability. Let us imagine that *Martha* has just arrived and wants to see what *John* and *Peter* are doing. For this purpose, the appointment book application allows other members' application interfaces to be replicated and represented remotely.

Martha's awareness subsystem will be in charge of inserting template tuples in the space so as to trace *John's* and *Peter's* pointers (see Fig. 8).

Since *John* and *Peter* are registered in the system and their activity can be observed, their *Awareness* subsystems are continuously inserting their mouse pointer coordinates (see Fig. 9) in the space. This kind of tuple lifetime is very brief as the information they provide is very punctual. It does not matter whether *Martha's Awareness* subsystem captures all *John's* or *Peter's* mouse pointer coordinates.

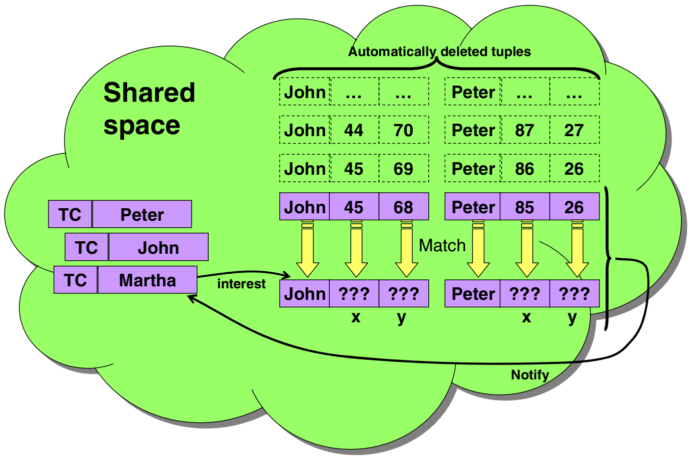


Fig. 9. Remote mouse pointer tracing

5 Physical Architecture and Deployment

JavaSpaces allows several spaces to be "running" on the same or different machines. This means that the tuple space need not be allocated on a specific machine and may also be distributed and partitioned between different sites. In any case, all the tuples in all the spaces form a single logical space of data. The space partitioning also allows pieces of information and tuples used by certain services (e.g. *Awareness* subsystem) to be allocated on certain machines depending on system performance. The way a space client application is "advised" about the entire tuple space distribution and the space distribution itself are beyond the scope of this work.

Each user has a replica of the application, i.e. the one he/she interacts with. In order to represent the behavior of other group members remotely, it is only necessary to replicate the events that the space notifies in the remote user interfaces that every user maintains for each user he/she is observing. This way of programming remote state replication is particularly efficient since a user need not send his entire state by means of complex data structures or objects; this state is instead defined separately on the basis of very short messages announcing the occurrence of certain events.

Current implementation has been carried out by using standard internet technology (web browsers and servers, Java, etc.). Fig. 10 shows how client applications and the space are not necessarily placed in any particular network node. The clients and services can be allocated in the same or different nodes. Clients interact with the system by inserting information into the space or retrieving information from it without being aware of where the tuples (or services) are.

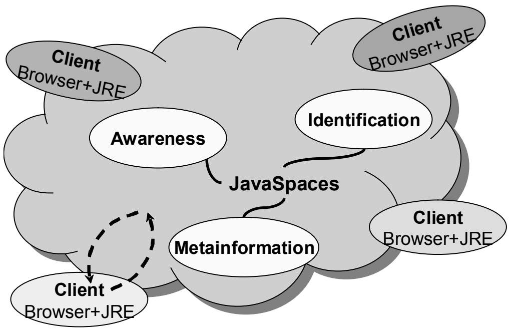


Fig. 10. Abstract view of the system and deployment

6 Related Works

The CSCW and groupware community have developed in the last years several experimental collaborative systems. Most of them have been developed for particular applications; this fact requires a great effort in implementations. Others have identified basic services for groupware systems developing toolkits to build these applications. For example, Groupkit [24] provides a component library for building multi-user interfaces. The main problems of this kind of proposal are:

- interfaces cannot interoperate between them, and
- difficulty of adaptation to the user's needs.

Pounamu [18] is a collaborative editing tool for software system design that permits work to be carried out both synchronously and asynchronously. It is built in Java using RMI and can be installed using different architectural designs, although it is limited to peer-to-peer collaboration scenarios.

EventHeap [16] provides a software infrastructure for interactive workspaces by means of an extension of TSpaces [32]. In [26] a flexible notification framework for describing and comparing a range of notification strategies is introduced; it can be very useful to guide the design of notification components. NSTP (Notification Service Transfer Protocol) [22] is a basic service (no semantics) for sharing state in synchronous groupware, therefore, it abstracts out the problem of state consistency from any application. The mechanisms implemented in these three proposals are similar to the presented in this paper, but in our case, the emphasis is mainly on the methodological context for these implementations (global and abstract architectures, methologies, …) instead of technological one (failure isolation, algorithms, consistency...).

MARS-X [4] and XML-Spaces [30] are coordination models that extend Linda model features using XML document properties for information sharing. Another architectural pattern for web services based on the Linda model (which also uses XML documents) is proposed in [1]. None of these addresses remote event replication in remote interfaces.

7 Conclusions and Future Work

Technological and social aspects influence human collaboration; being aware of other group members' activity diminishes the risk of work duplication and inconsistencies in proposed tasks deriving from their concurrent accomplishment. This paper proposes an implementation for fulfilling functional requirements related to group awareness. It is part of a conceptual and methodological framework.

The state of the system is described by means of tuples in a space; replicas of the same groupware application and interactions between subsystems can be technologically coordinated exchanging tuples through spaces instead of communicating directly. Another benefit is the temporal and spatial dissociation inherent to this paradigm since a space and its tuples may remain in the system, even after the process that created them has ended. At this level, implementation and deployment issues are abstracted thanks to the coordination model that eases the building of distributed applications.

Whether it is distributed or not, a tuple space (such as the one our collaborative appointment book interacts with) imposes no restriction about the computer where it must be placed. This enables us to fulfill certain non-functional requirements related to profitable transparency characteristics in distributed systems: localization, access, replication, concurrency and scalability. This degree of transparency simplifies architecture development and facilitates the application of concrete guidelines in groupware systems building. As outcome of this research work a hybrid collaboration architecture [6] has been proposed; application is replicated in each network node, and control is logically centralized (actually distributed or centralized depending on the tuple space implementation). The main advantage of this approach is that users keep local replicated copies of other participant interfaces without sending the objects which define them (the interfaces).

There are other benefits of the proposed solution made apparent from the experience acquired in the development of a real example. Although performance analysis has not been carried out yet, apparent results obtained executing the collaborative appointment book application on internet seem promising.

By way of future work, we have started to develop a graphic component library (toolkit) for building groupware applications following the hybrid architecture mentioned above. The aim is to include and encapsulate the implementation philosophy described in this work within these graphical components in order to simplify the subsequent groupware systems development. In addition, portability in the current implementation is Java-dependent, that is, the applications which interact with the space must be programmed in this language. Some of the related works avoid this handicap using markup languages (XML) in the definition of the tuple space and thereby, hiding implementation issues and promoting interoperability.

Acknowledgements

This research is partially supported by the R+D project TIN2004-08000-C03-02 of the Spanish MCYT.

References

1. Álvarez, J., Bañares, J.A., Muro-Medrano, P.R.: An Architectural Pattern to Extend the Interaction Model between Web-Services: The Location-Based Service Context. M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 271–286, 2003
2. Architecture Working Group: Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std 1471 (2000)
3. Beaudouin-Lafon M.: Computer Supported Cooperative Work. Université Paris-Sud, France, John Wiley & Sons (1999)
4. Cabri, G., Leonardi, L., Zambonelli, F.: XML Dataspaces for Mobile Agent Coordination. In 15th ACM Symposium on Applied Computing, pages 181–188, 2000
5. Carriero, N., Gelernter, D.: Linda in Context. Communications of the ACM, Vol. 32, No. 4, 444-458, April 1989
6. Chung, G., Dewan, P.: Towards Dynamic Collaboration Architectures. Proc. ACM CSCW'04, 1-10
7. Ellis, C.A., Gibbs, S.J., Rein, G.L.: Groupware: Some Issues and Experiences. Communications of the ACM, Vol. 34, No. 1 (January 1991) 38-58
8. Garrido, J.L., Gea, M.: Modelling Dynamic Group Behaviors. In: Johnson, C. (ed.): Interactive Systems - Design, Specification and Verification. LNCS 2220. Springer (2001) 128-143
9. Garrido, J.L., Gea, M.: A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling. In: Forbrig, P. et all (ed.): Interactive Systems - Design, Specification and Verification. LNCS 2545. Springer (2002) 16-28
10. Garrido, J.L., Gea, M., Rodríguez, M.L.: Requirements Engineering in Cooperative Systems. Requirements Engineering for Sociotechnical Systems. IDEA GROUP, Inc.USA (2005)
11. Garrido, J.L., Padereswki, P., Rodríguez, M.L., Hornos, M.J., Noguera, M. A Software Architecture Intended to Design High Quality Groupware Applications. Proc. of the 4th International Workshop on System/Software Architectures (IWSSA'05), Las Vegas (USA), June 2005
12. Gelernter, D., Carriero, N.: Coordination Languages and Their Significance. Communications of the ACM, Vol. 35, No. 5, 96-107, February 1992
13. Grudin, J.: Groupware and Cooperative Work: Problems and Prospects. In Baecker, R.M. (ed.) Readings in Groupware and Computer Supported Cooperative Work, San Mateo, CA, Morgan Kaufman Publishers (1993) 97-105
14. Gutwin, C., Penner, R., Schneider, K. Group Awareness in Distributed Software Development. Proc. ACM CSCW'04, 72-81 (2004)

15. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999)
16. Johanson, B. and Fox, A., "The Event Heap: A Coordination Infrastructure for Interactive Workspaces". In Proc. of the 4th IEEE Workshop on Mobile Computer Systems and Applications (WMCSA-2002) Callicoon, New York, USA (June, 2002)
17. Macaulay, L.A.: Requirements Engineering. Springer (1996)
18. Mehra, A., Grundy, J., Hosking, J.: Supporting Collaborative Software Design with a Plug-in, Web Services-based Architecture. Proc. of the ICSE 2004 Workshop on Directions in Software Engineering Environments, May 2004, Edingurgh, Scotland, IEE Press
19. Object Management Group: Unified Modelling Language (UML) 2.0 Superstructure Specification (OMG), August 2003. Ptc/03-08-02, 455-510
20. Papadopoulos G.A. and Arbab F.: Coordination models and languages. Advances in Computers, 46, Academic Press (1998)
21. Paternò, F.: Model-based Design and Evaluation of Interactive Applications. Springer-Verlag (2000)
22. Patterson, J., Day, M., Kucan, J.: Notification Servers for Synchronous Groupware. In Proc. of the ACM Conference on Computer-Supported Cooperative Work (CSCW'96) 122-129
23. Phillips, W.G.: Architectures for Synchronous Groupware. Technical Report 1999-425, Department of Computing and Information Science, Queen's University, May 1999
24. Roseman, M. and Greenberg, S.: Building Real Time Groupware with GroupKit, A Groupware Toolkit. ACM Transactions on Computer Human Interaction 3(1), (March 1996)
25. Schlichter J., Koch M., Burger M.: Workspace Awareness for Distributed Teams. In: W. Conen, G. Neumann (eds.), Coordination Technology for Collaborative Applications, Springer Verlag, Heidelberg (1998)
26. Shen, H., Sun, C.: "Flexible Notification for Collaborative Systems" In Proc. of ACM 2002 Conference on Computer Supported Cooperative Work (CSCW'02), New Orleans, USA (2002) 16-20
27. Sommerville, I.: Software Engineering. Addison-Wesley (7th ed.) (2004)
28. Sun Microsystems: Jini Distributed Events Specification. Version 1.0. Available on-line at http://java.sun.com/products/jini/2.1/doc/specs/html/event-spec.html
29. Sun Microsystems: JavaSpaces Service Specification. Version 2.2. Available on-line at http://java.sun.com/products/jini/2.1/doc/specs/html/js-spec.html
30. Tolksdorf, R., Glaubitz, D. XMLSpaces for Coordination in Web-based Systems. Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 06 20 - 06, 2001
31. van der Veer, G., Lenting, B., Bergevoet B.: GTA: Groupware Task Analysis - Modelling Complexity. Acta Psycologica, 91 (1996) 297-322
32. Wyckoff, P., McLaughry, S., Lehman, T., and Ford D.: TSpaces. IBM Systems Journal, 37(3): 454-474, (1998)