Coordination and Tailorability Issues in the design of a Generic Large Scale Groupware Platform

Henrique João L. Domingos José A. Legatheaux Martins

{hj,jalm}@di.fct.unl.pt

Departamento de Informática - Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

PROPOSAL PAPER FOR THE WORKSHOP on
TAILORABLE GROUPWARE: ISSUES, METHODS and ARCHITECTURES
ACM - GROUP'97, Phoenix, Arizona 1997

Abstract

Tailorability or adaptability is commonly accepted as a fundamental property of modern and flexible CSCW systems. In order to match the differences in organizational structures, these systems have strong requirements in terms of dynamic reconfiguration and adaptation.

This work is concerned with the design and implementation of a tailorable generic distributed system platform for large scale CSCW. We argue in favor of an approach for the tailorability support, focusing different perspectives and emphasis according to different support levels. This approach can help in a better understanding of different structuring levels involved in the support of adaptability. We discuss and ilustrate how these different perspectives are present in the proposed integrated framework and we exemplify the usage of a component model (JavaBeans) as a candidate to build software components for groupware, dealing with the different recognized structuring levels of the tailorability support.

Keywords*: CSCW/Groupware, Tailorability, Flexibility, Coordination, Large Scale Distributed Computing Systems, Object-Oriented Programming, JAVA, Software Components, JavaBeans.*

1. Introduction

Tailorability and adaptability are currently referred as fundamental properties of CSCW environments. Different authors are emphasizing different approaches for its support. In some perspectives, the need of flexible supports for static and dynamic configuration/customization is referred as a crucial requirement. In other approaches, the use of integrated models supporting the interoperability with existent well-known distributed services is mentioned as a fundamental issue. Finally, other contributions emphasize the adoption of object-oriented design methodologies and the need of open groupware architectures based on the composition of software components.

In our viewpoint, all the above perspectives of tailorability are relevant. What is needed is a way to highlight what is important at what approach level. This paper is a contribution towards this direction. We identify two main perspectives corresponding to different levels of dealing with tailorability issues. These perspectives are related with two fundamental directions in the materialization of a generic and integrated distributed system platform and framework for the support of large scale CSCW environments.

Paper Organization. Our contribution for a structured approach of the tailorability supports involved in such a framework is presented as follow: **section 2** presents an overview of the two main recognized tailorability perspectives; **section 3** is a brief introduction to large scale distributed collaborative systems (LSDCS), whose properties are subjacent in the design of a framework for the development and operation of scalable groupware applications; **section 4** maps the recognized tailorability support levels to the different framework support layers; **section 5** develops the architectural model of the proposed framework; **section 6** presents an example of how tailorable components can be implemented with flexibility in terms of granularity and heterogeneity; finally **section 7** presents a brief summary as well as some conclusions.

2. Tailorability in groupware

The tailorability support for groupware can be understood in two main perspectives: (1) a perspective where the tailorability is a property closely related with the "openess", "reconfigurability" and "interoperability" of tools in generic and open CSCW environments and (2) a perspective where the tailorability is a property related with a software engineering perspective for groupware development.

Tailorable groupware in open CSCW environments

The problems of developing tailorable groupware in CSCW environments, are compounded by the strong inter-relationship between the implementation design of each application and the organizational structure in which it will operate. In this organizational perspective, those applications are not used in isolation but are an integral part of other kind of collaborative activities which take place within the organization where they are used.

In this environment, provision of the tailorability support in CSCW must be realized as an open architecture based on adjustable component-tools[1] that must be implemented and/or configured (or customized) accordingly with adequate coordination and management policies, in a generic way. This architecture should allow a multiplicity of flexible approaches and paradigms to co-exist at different levels, ex.:

- remote **vs.** local interaction support
- synchronous **vs.** asynchronous (or disconnected-based) group oriented interactions
- personal/autonomous **vs.** group/dependent collaborative work
- formal **vs.** informal group procedures and collaborative processes
- volatile **vs.** persistent data objects shared in the context of applications
- open activities participated in a voluntary based **vs.** closed activities with participation control
- different roles played by different users in different specific activities (applications) **vs.** peer-based responsibility models
- well-defined temporal activities/tasks **vs.** permanent organizational procedures
- well defined goals with fixed deadlines **vs.** on-going work and sub-activities results

This can only be achieved by the provision of common coordination components and basic mechanisms supported in the environment, while applications concentrate on task-specific functions and in the modeling of specific workgroup methodologies. This approach must allow several models of cooperation/coordination to cohexist in an adjustable way.

Tailorability under the software engineering perspective

In the literature, many approaches for tailorable groupware are based on the advantages of using design models for groupware architectures based on the composition of software components [2]. To tailor a specific application or tool, the components are composed and its specific implementation is adapted to meet specific requirements.

One of the problems involved in this perspective is the correct analysis of hierarchic dependencies between the components. In a bad structured model, the dependencies can cause problems of global adaptability when each component is modified individually.

Other difficulty is a clear-perspective of the abstractions provided at different levels of the composition hierarchy. Many groupware applications are very complex in terms of the structure and interdependencies between functional requirements. Furthermore, the kind of building blocks that must be considered as possible candidates for basic collaborative components, is something that it is not clear (yet) for the groupware research community.

3. A brief perspective on Large Scale Distributed Collaborative Systems (LSDCS)

---

1 We will use the term "component-tool" as a term describing a component granularity related with a self-contained functionality regarded at an organizational viewpoint. A component-tool can correspond to a simple tool, which must be integrated as a collaborative application implementing a specific methodology for a collaborative task in the context of a more generic and complex collaborative group activity.

2 Software components are, in this context, components designed and implemented by component-oriented programming environments and supported by component architectural models. Thus, software components correspond to a software engieneering perspective of components.

Generic platforms for CSCW (ex.: [Knister92][Roseman92]), and those specifically based on LSDCS supports (ex.: [Cosquer96], [Kindberg 96], [Dom97]), require high levels of integrability (coexistence with traditional standalone applications and services), interoperability (execution and interoperation in heterogeneous platforms), maintainability (easy maintenance/upgrading in functional requirements), customization (to adequate the systems to operational needs) and adaptability (to accommodate easily the specific functional requirements to the user-needs). The "*large-scale factor*" also requires *fault-tolerance* and *high availability*. Moreover, as in traditional distributed systems, properties like: *openness*, *standardization of components* and *interprocess communication protocols* are also essential criteria.

The design of an integrated large scale CSCW is a case-study where the two above main perspectives of tailorability must coexist and must be regarded in a complementary way.

The structural organization of the collaboration workspace requires the definition of a coordination model based on customization, configuration and management criteria of collaborative workgroup activities. These issues deal with the need to warrant the base conditions for group cooperation in a productive and creative way and to fulfill the organizational convergence of different group activities. The flexibility needed in the support of such facilities require the definition and adoption of efficient coordination models in which each specific groupware application is used as a specific methodology to develop a simple task, in the context of a more complex workgroup activity.

In what concerns the software engineering perspective, a large scale CSCW system is a case-study where the adoption of an open-architecture based on the composition of collaborative-oriented software components is an interesting approach presenting obvious advantages. Due to the impact of scale, these systems exhibit considerable complexity at different support levels. We structure these supports in four fundamental levels:

> **Communication/replication**: at this level we emphasize the need of reliability warranties for fault-tolerance and flexible group oriented interaction models based on multi-synchronicity criteria. The support must provide the basic abstractions for the materialization of object replication models;
>
> **Group collaborative memory**: this support deals with replication models to implement scalable object-repository services. The support of persistent collaborative workspaces where collaborative objects are stored with high availability is the main goal.;
>
> **Group coordination and management**: deals with coordination control models and group oriented management services. This support is very relevant to achieve the success of collaborative workgroups, providing the basis for the management (detection and solving) of conflicts and dependencies;
>
> **Components integration support**: this support provides facilities allowing the integration between different groupware applications as well as between groupware applications and wide-used tools (e.g., video and audio communication tools, awareness-control services and other information dissemination supports).

Each of the above levels, has specific emphasis and different focus in the way tailorability may be supported.

We emphasize that a fundamental issue directly related with tailorability requirements in collaborative environments is the flexibility for the support of different coordination methodologies and specific collaborative work management. Although many authors and projects deal with the support of coordination issues in specific groupware (e.g. workflow and group-decision support systems) only few publications focus on the coordination support in a more generic perspective. This perspective, however, must be considered for the materialization of common coordination principles of collaboration in terms of the computational support. This perspective is very relevant to establish the fundamental difference between "enabling the basic support for cooperation" (which is the main focus on group oriented communication tools and group interaction supports) and "supporting effective and productive cooperative work" (which must be understood in the perspective of the productivity criteria).

4. Supporting tailorability in large scale CSCW

Our focus on tailorability requirements is concerned with the flexibility provided by the architectural model of generic computational supports for groupware. We are not concerned by organizational structures or sociological human factors.

As we have introduced in section 2, we have identified two main directions by which tailorability issues can be considered:

- The first direction is related with system configuration, customization and management. It also involves the adoption of standard component-tools and integration facilities. It is, basically, a way of defining, structuring and materializing coordination computational models in the context of large scale CSCW activities, supporting those facilities.
- The second direction (corresponding to the software engineering perspective) deals with the adoption of tailorable software components, as collaborative oriented building blocks, reused (adapted, adjusted and composed) in the context of the programming environment chosen to build specific collaborative applications.

In our viewpoint, the two above directions are related and only reflect two different granularity levels and adaptation time. We will describe in more detail the two identified directions in the characterization and materialization of a generic groupware framework for large scale CSCW.

4.1 Tailorability as a property of CSCW management and coordination

This form of tailiorability is based in the way how collaborative work coordination is supported. Thus, its support is closely related with the definition of a coordination model for the framework, which must support the organization of collaborative workspaces, the structural organization of collaborative workgroups asserting group-convergence and productivity from the organizational viewpoint.

The current generation of CSCW applications provide diverse models and mechanisms intended to support a particular "artificial" cooperative activity or a class of activities modeling specific methodologies. In general, each groupware application, implements its own specific coordination support and model. These applications are often unaware of the existence of other applications in the same environment and provide no (or at most only few) interoperable mechanisms.

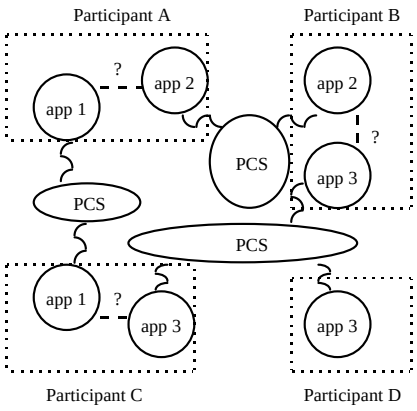


Fig 1A. Confined Coordination Spaces

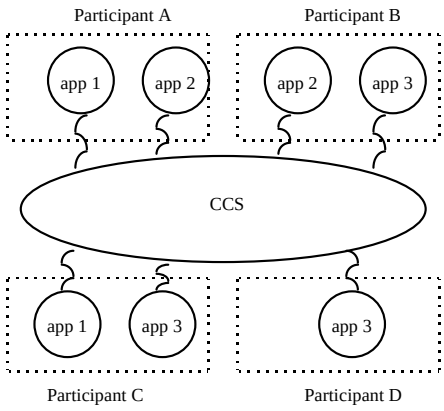


Fig 1B. A Global Coordination Space

Using these applications separately, users of each CSCW application are presented with a particular model and are confined to different closed coordination worlds (fig.1A). An integrated and global coordination vision relating different parallel tasks do not exist. Thus, the user looses the perspective of the relevance of each partial task in terms of the whole cooperative goal. By other words, there is not an unique and global common goal vision but only partial perceptions of group activities and interpretations of each task relevance. All this, in detriment of productivity, creativity and group members motivation.

The role of an integrated system environment in which different groupware applications are supported becomes a crucial factor for the effectiveness of CSCW. A fundamental aim of such environments is to provide interoperability between a large variety of possible applications in the context of the same global coordination workspace (fig.1B). Applications must work in harmony rather than in isolation of each other. In addition, it is absolutely necessary that the system coordination facilities at different levels provide the necessary mechanisms to make adjustments based on interoperability criteria.

4.2 Tailorability as a property of flexible groupware design and implementation

As has been said, this direction deals, basically, with the design and adoption of tailorable software components as collaborative oriented building blocks useful for groupware development (ex. [Batteram 97]).

This perspective has been motivated by design methods of software engineering directly inspired by the traditional mechanical and electronics engineering design processes. A software system based on independent functional components can be understood as an architecture of components. To implement these architectures, the programming environment must provide facilities for components reuse and composition control, optimizing the development or adaptation processes . At same time, the approach presents important advantages in terms of quality and robustness of the global architecture which can be improved because well-known standard and well-tested software components tend to be less prone to errors.

On the component level, such systems can be described in terms of:

- A specification language for components composition
- The definition of uniform API's to support the interaction between components
- The underlying component model and its interface

The composition of basic components is often provided by specialized application builders. In many industrial products, these application builders are visual environments[3] . Basically, application builders provide a specification language with constructs describing the bindings between different components. They also offer facilities for components configuration with adjustable parameter settings. These supports implement the APIs allowing the interaction between components, hiding from the programmer the underlying implementation model.

The advantage of adopting software components composition in the development of a groupware generic platform, is that we can reason on each specific application-level abstraction. In fact, this will be possible when the recognition, acceptance and validation of basic specific-collaborative components implementing reusable functions will be stabilized (situation from which we are currently far from). This is the situation that occurred with other areas of software engineering (ex: GUI-design).

5. The architecture of a tailorable framework for large scale CSCW

As we have described in the introduction of the paper, we are focused on building an integrated and generic platform and framework for large scale CSCW. In this section we will describe this framework, discussing the different tailorability issues involved. This framework is fully implemented in JAVA and adopts the properties and characteristics of this language as starting points to provide tailorability. We will describe how we deal with adaptability at the different levels of the framework.

- Configuration/customization and integration criteria: we adopt an open layered architecture for the framework and we have defined a flexible coordination model for collaborative work.
- Integration based on well-defined APIs provided by the framework basic support services: we adopt APIs for basic group-oriented communication services and support for multi-synchronous group interactions, as well as scalable object replication.
- Components architectural model: we adopt a components model to support the adaptation and mutual composition of specific collaborative oriented building blocks that are going to be gradually developed. In this direction we will

---

[3] Examples of programming environments offering application-builders to compose and integrate applicational-components are, for ex.: Microsoft Visual Builders Visual C++/ or Visual Basic which adopt the OCX components model, Visual J++ that adopts the JAVA language native facilities or application builders supporting the JavaBeans model, ex.: "IBM's Visual Age for Java", "SunSoft's Java Workshop", "Lotus Development's BeanMachine", "Symantec's Visual Café" and many others [JavaSoft97].

discuss the adoption of Java-Beans as a possible support to implement component-oriented functional building blocks. Section 6 shows a specific example of this strategy.

5.1 The coordination model

In order to structure our platform[4] we have defined a coordination model in which workgroup participants are organized in *persistent collaborative sessions*. A collaborative session acts as a global coordination shared space corresponding to a specific collaboration domain. A session takes place when a set of users, sharing a persistent workspace, cooperate to achieve a stated goal previously selected and commonly accepted. Each session corresponds to a well-known notion of *cooperative workgroup*.

The participation in a given session can be *reserved (*when the session is *closed* and the group members are previously registered by an administration authority in a registration service) or *free* (when the session is *open*). In both cases, session-binding mechanisms are provided by the coordination support. These binding-mechanisms allow users to start dynamically the applications and tools selected by the session manager.

Users act in each session with possible specific roles. A role represents a "social responsibility" of each user. In terms of the computational model, this responsibility is "mapped" in a set of privileges/authorizations. The process of session admission (authentication and session logon) determines simultaneously the initial role with which the participant interacts with the collaborative workspace. Subsequently, users can act with different roles (acquiring different access permissions in a dynamic way).

Each individual user should be allowed to participate simultaneously in different sessions (being a member of different workgroups), however, in specific moments, she/he is normally concentrated on just one. For coordination purposes, the workgroup participation has two levels of relevance: the group organizational structure defined statically at session level configuration and the dynamic group membership participation, which is variable in time and is monitored by means of group awareness control mechanisms. The first level represents the potential workgroup organization and, in fact, is more relevant in closed sessions. The second level represents the dynamic group membership status and is relevant for coordination awareness purposes.

Group organization and configuration is supported by a registration and authentication service (a directory service). Each user entry can include all the relevant social information (ex: name, nickname, e-mail address, user public key, user web homepage, pager/phone number, social role, social identification , …).

The most relevant fact in terms of tailorability is that each collaborative session is not supported by an unique complex monolithic application, but by a collection of collaborative tools. From the software-engineering perspective this multi-tool approach for the session has important advantages over the approach of providing all the functionality (coordination and communication) from scratch, in every application.

From the practical viewpoint, the session coordination level is implemented by means of a set of coordination rules and generic information. When each user joins to the session, she/he will dynamically load a special application (known as a "session-manager"), that manages the coordination rules and other coordination/management information and provides facilities for dynamic loading of the other collaborative tools to be used during the session. The session coordination/management information is shared by those tools. In addition, each cooperative tool shares, particularly, specific data-objects, and possible specific coordination and awareness control information, in a sub-set of the global session workspace.

This model is tailorable in the sense that while in a cooperation, users tend to plan, phase and divide complex tasks in smaller sub-tasks. Each sub-task has a specific cooperative goal and a specific work methodology materialized by particular applications. In terms of the computational model, work methodologies are represented by specific tools or applications. These tools can be dynamic configured (or substituted) for each session (by an user or users acting with a special role for this). In such a model, the participants adopt different *group-oriented collaboration-aware tools and applications* in the context of the same coordinated session workspace. The tasks developed with different tools can be

---

4 The structuring model discussed in this section is beeing implemented in the context of a research prototype called Dágora ("Distributed ágora"). DÁgora is a generic object-oriented CSCW platform for the support of flexible and scalable groupware applications. Currently, the Dágora support provides flexible reliable group-oriented communication services for adaptive synchronicity (supporting synchronous as well asynchronous group interactions or disconnected collaborative work in a complementary way). The system also implements a scalable replicated object-storage system, providing the basis for the support of persistent workspaces and data-repository services.

accomplished in parallel or in sequence, to achieve the expected common cooperative goal established for the global session.

5.2 Organization of the workspace

Fig.2 represents the workspace topology for a large scale collaborative environment supported by the proposed framework, as we will describe below.

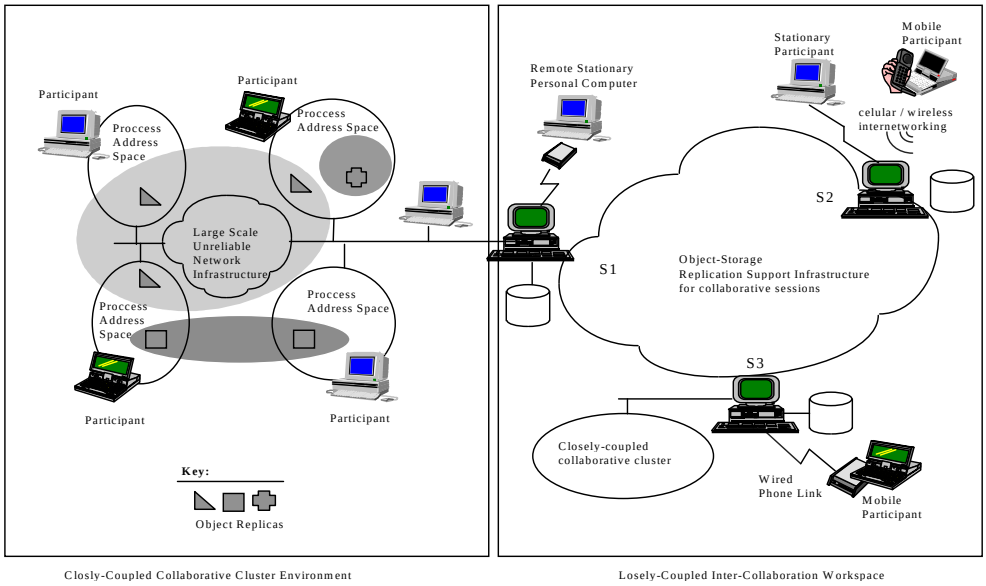


Fig.2 Large scale workspace reference topology for the framework

As presented we consider different domains of cooperation and coordination:

Closely coupled collaborative clusters. These clusters are collaborative domains supported by an internetworking infrastructure that allows synchronous "soft-real-time" interactions among closely-coupled collaborative groups. The fundamental criteria to define such a cluster are: its potential responsiveness characteristic (comparable in magnitude order with local response times), low latency and good quality of service provided by the communication facilities. Closely coupled collaborative clusters allow a good level of interactivity, good support provided by informal communication channels (audio/video tools or even periodic face-to-face meetings), limited dispersion of groups (numbers of groups per collaboration and number of participants per group) and homogeneous backgrounds of the participants involved.

Large scale inter-collaboration workspace. This space is supported by an open architecture based on a scalable object-storage replication infrastructure shared by different collaboration sites. A wide area replicated support can be provided by a distributed group of replicated servers (ex: S1, S2, S3) managing a persistent global data repository. In each site, a local Object-Storage Server provides access to the group-collaborative memory of each collaborative session.

Each local replica provides the support for asynchronous interactions and for the dynamic (re)integration of mobile and partially connected participants. Each server also assures the basic coordination and management facilities for the currently maintained collaborative work sessions. For this propose, it provides administration and configuration tools and session-management services. In summary, this infrastructure offers high availability and is the essential support of the dynamic reconfigurability and scalability criteria for large scale cooperations taking place in the inter-collaboration workspace.

5.3. The structuring model of the framework and the different perspectives of tailorability involved

Fig.3 represents the general model of the framework and the different levels where tailorability is relevant.

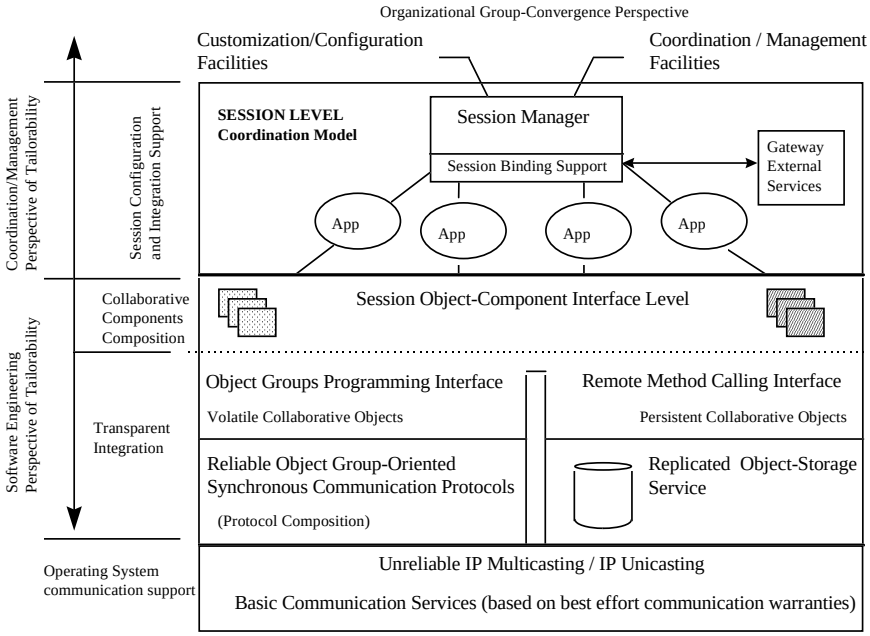


Fig.3 Generic structuring model for the framework and perspectives of tailorability

5.3.1 Base support of the framework for large scale collaborative environments

Support of peer-synchronous group cooperation for closely-coupled collaborative clusters. Synchronous cooperative applications running on different closely-coupled collaborative clusters are organized as specific synchronous sub-sessions scheduled in the context of each persistent collaborative session.

Synchronous groupware applications are highly environment sensitive. User actions are strongly driven by unpredictable events. The system support should provide active upcalls representing reactive event notifications driving the awareness control needs for different kind of applications. These requirements are facilitated in terms of the programming support by providing peer-synchronous collaborative-oriented object-group abstractions on top of reliable group-communication protocols stacks.

For this support, new requirements (not necessarily found in general settings provided by well-known group-communication technology [Birman 94], [Renesse 94], [Cheriton 85], [Kaashoek 91], [Amir 92]) are raised, namely: short or immediate responsiveness, short event-notification times and implementation of multi-consistency object-replication control criteria. A more detailed description about this support is available in [Simão 97]. The implemented group-communication protocols are specifically tailored for synchronous cooperative interactions. In this, we include the ability to use the semantic knowledge of each application to choose the appropriate semantics of certain group communication protocols available. Other parallel communication channels, (e.g., video, sound and other bilateral communication facilities) can be used as *ad-hoc* synchronization mechanisms avoiding to pay the "price" and overhead of using a heavyweight protocol in certain cases. These considerations are addressed by a special group-communication support service in the framework, which provides basic object-group communication protocols and group membership management control.

This support is provided "atop" of a stackable and extensible runtime machine supporting dynamic compositions of multi-semantics and reliable group-oriented communication protocol layers, as in [Renesse 94]. These layers are all implemented in JAVA. Basically, this support implements fault-tolerant group-oriented communication protocols, enforcing adequate consistency criteria added to the only simple "best-effort" guarantees offered by the basic IP multicast protocol.

This is one important direction of our work, consisting in actively devising new fault-tolerant lightweight object-group communication protocols and group view-membership management services, specially tailored for peer-synchronous groupware. The protocols implemented take an optimistic replication strategy since for many applications they better match the recognized requirements of groupware. As the protocols are available "atop" of a lightweight stackable JAVA runtime "machine", the framework provides facilities to allow dynamic loading of protocol stacks, "on demand", by different cooperative peer-synchronous JAVA applications. This approach provides flexible options for fully replication object-oriented groupware and allows the support of heterogeneity.

To develop a synchronous application, the application state and the execution context are replicated in each user workstation. Fully replication promotes high availability, fault-tolerance and load-sharing in the overall system. However, it must be granted certain degrees of consistency between the replicas used in each moment by different sites in the context of different applications. As this is a complex question, the system offers facilities to deal with the consistency control, isolating groupware applicational programmers from such complex details.

Consistency control and group binding services are offered, transparently, as part of the object-group abstraction provided by the group communication sub-system. Starting from a session manager application it is not difficult to bind to a specific synchronous application just downloading the application as a JAVA class, as in the WEB. Parameters and attributes managed at session coordination level can be obtained as input arguments for application session binding. Each application can use the existent coordination object-components to manage the coordination information managed in the object storage infrastructure in a persistent way. This support is provided by means of a remote method calling interface. We adopted the native RMI (remote methods invocation) support of JAVA for this purpose.

Asynchronous cooperation and disconnected work

The basic support for asynchronous/disconnected collaborative interactions is composed by the following facilities provided by the Object-Storage infrastructure:

- Objects model for the persistent collaborative data repository supporting the group collaborative memory. This support allows several users to modify the same data concurrently, not restricting their actions, besides usual access control mechanisms defined by the coordination rules enforcement;
- A weak consistency control system for persistent object replication in large scale.
- The support to establish coordination information and binding functions for collaborative sessions and applications;
- A versioning control mechanism supporting asynchronous updates and "disconnected" operations. With this support, concurrent modifications on resulting shared data-objects are conjugated, enabling type specific resolution and detection of conflicting updates.
- Log-based management information about user activities, data-objects' updates and conflicts detection/resolution. This information is reusable for basic asynchronous event notification services and awareness control purposes.

A more detailed description of the object-storage infrastructure and the replication support is available in [Simão 97].

5.3.2 Perspectives of tailorability support within the framework

Coordination/management control perspective

This perspective of tailorability deals with configuration and management facilities provided at the session level of the coordination model. At this level, component-tools implementing the coordination model provide facilities for its re-use as session coordination components. These tools extend a generic session object providing static and dynamic configured coordination settings and parameters and other useful management information in the context of different applications used in each session. Most of this information is maintained in a persistent way by the object-storage infrastructure. Thus, the tailorability perspective of the framework is oriented for customization and configuration needs.

Specific support for integration with other services and/or commonly-used communication tools is provided by means of session gateways, which can be managed as any other coordination component-tool.

By means of the session manager service, each participant has automatic access to the different applications used in each session. The session manager acts as a kind of collaborative shell, in which applications are launched inheriting environment variables. This session configuration is established by an user (or several users) acting with the role of creating, modifying, and removing sessions. When a user joins a current registered session, she/he will use by default, the current configuration and coordination information globally configured for that collaborative session.

Software Engineering Perspective

Closely-coupled cooperative applications based on group peer-synchronous interactions adopt a peer-synchronous object-group programming interface. This API supports group-oriented synchronous communication. Asynchronous loosely-coupled collaborative applications use a standard remote method invocation (RMI) interface, interacting with the support provided by the replicated object-storage infrastructure. Multi-synchronous applications requiring adaptive synchronicity can use both supports (including disconnected asynchronous interactions) in a complementary way.

Specialized libraries providing specific collaborative and coordination object components for the applications can be materialized starting from these base APIs. As described above, the idea is that these objects must implement reusable standard and specialized adaptable building blocks for specific functional requirements in different applications. From the software engineering perspective of tailorability, the programmers use the fundamental abstractions of the base framework support as starting points for the definition of an open components based architecture.

Different session manager applications can be implemented, basically, as any other asynchronous application, using, in this case, specific coordination components.

The major trend in a groupware engineering perspective is in fact the definition, validation and acceptance of which kind of groupware software components must be provided. This deals not only with which components but also with the concerned abstractions at different levels of applications composition. In our viewpoint, this is one of the more essential problems that groupware research must clarify in the short-run.

The other direction in the software engineering perspective, is the need of specific programming environments offering a component-model architecture useful for groupware design. This support is essential, to allow all the kind of transformations involved (even those materialized dynamically during the multi-use of the final applications). These possible transformations can occur in different ways: changes introduced in parameters defined at each component level, modifications in the structure of the composition of different components, changes in the internal implementation of each specific component and dynamic substitution of similar functional components, which must provide the same interface, but implement the same internal functionality in a different way.

This support is based on the assumption that the composition language employed in the description of the component architecture must provide (ideally) a standard approach in the following facilities: (in order to be supported by different application builders):

- Configuration: which relates with possible adjustments based on parameter settings of a specific software component (e.g., a label in a simple button, the number of scrolling bars in a multi-scroolbar component, a synchronous group membership control component setting a failure suspector on a group-membership protocol, multi-purpose event-notification components, etc.);
- Composition: which deals with the support of connectivity between components, describing possible complex hierarchical composition models (sub-compositions in more and more complex components not individually composed in the final application);
- Implementation: this deals with the implementation of new components (from scratch). In typical environments an object-oriented programming language is used. In our case, standard supports for distribution, heterogeneity, dynamic object binding and dynamic loading of remote classes are of great interest

6. Developing specialized components for the framework

In order to promote tailorability at the level of tools programming, we need a component model. By adopting the JAVA language, properties useful for large scale environments (support for heterogeneity and code-shipping), are already present in our framework. We also need some form of flexible granularity for the components development. The components developed will be used as building blocks in composite applications. Probably, some kind of coordination components can origin regular self-contained applications, which may then be composed together at the

level of session coordination model. A specific coordination tool might live within a composite session-manager application as well as can be used as an independent application dynamic loaded from the session manager.

In conclusion, the components model should provide a "continuum" of granularity support in the component composition more than a sharp cutoff between composite applications and composite components.

These requirements are fulfilled by a software component environment like JavaBeans [JavaSoft97], which can be seen as a set of APIs which goal is to define a software component model for JAVA. These APIs provide facilities to create JAVA components that can be composed together into different kind of applications (or composite applications). The fundamental concept in the JavaBean model is the notion of *Bean*, which is defined as a reusable component which can be manipulated (eventually visually) in the context of an application builder tool. This means that a *Bean* component must be based on a consistent set of APIs which enforce that the *Bean* will work in the same way everywhere a JAVA runtime abstract machine is available.

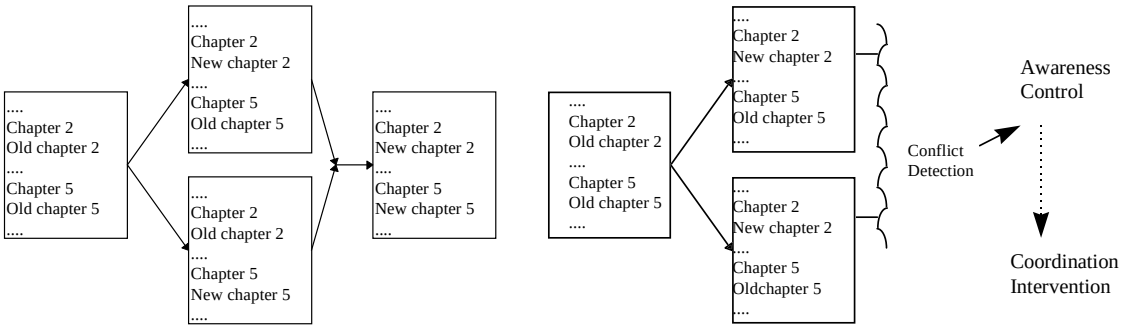
A tailorable notification component for asynchronous and disconnected collaboration

Below we illustrate a component approach with JavaBeans for a tailorable notification service building block. This component can be very interesting to structure and adapt different event notification needs related with the awareness control requirements in collaborative asynchronous and disconnected applications.

In the basic support for asynchronous collaboration, we use an object-oriented approach for the object-storage support, which fulfils the requirements presented for the group collaboration memory support because:

- Data types fit well in the object-oriented paradigm, because they are usually well-structured and have a well-known set of operations that can change their state. Thus, we can easily map data-types in class definitions. In general, despite being saved as unstructured files, data is well structured at application level (e.g., source programs, word processor output, LaTeX files, etc.). For example, a JAVA source file can be saved in an unstructured file, however, it can also be manipulated like an object with operations to insert/remove classes, methods, etc.
- As updates are applied by (logged) method invocations, it is possible to easily determine the contributions done in each work session. These contributions are recorded as the sequences of method invocations made by users. These sequences are replayed whenever wanted, thus making it possible to easily conjugate changes made by different users. For example, the Coda distributed file system [Kistler91] uses this approach successfully for directory management.

In order to gain access to objects managed in the replicated object-storage infrastructure, each application fetches from the best accessible replica its state and creates a local copy of it; future operations are performed locally without requiring communication with the object-storage. Update methods invocations are logged by clients, until later re-integration with an object copy located at the server adopted. Updates performed concurrently by different users are combined when logged updates are propagated to each server replica. For example, in figure 4 we have an object that implements an interface to manipulate a document type, and two users that concurrently modify two different chapters.



A - Automatically conflict solving B - Conflict detection for explicit resolution

Fig.4 Expected evolution of a collaborative editing session intervention
(The middle versions represent two concurrent modifications in the same document).

After modifying a chapter, the correspondent method invocation information is propagated to the object-storage server. When both updates reach a server, they will be reflected in the final document without any problem (automatically if there are non-conflicting versions or with an external coordination explicit intervention in the other case). Awareness control by means of a notification service component is fundamental, as a support for this explicit coordination.

There is a clear distinction between the base system and the implementation of specific component visible objects. The base system is reduced to a simple but efficient core of operations. Its main tasks are: to offer high-availability to clients; to propagate clients' updates to servers and to propagate objects' updates among all servers, replicating each object. The separation between the base system and object classes implementations is very important to optimize the base system and to enable the easy implementation of new object classes with different and new semantics, without modifications of the object storage infrastructure. This is a similar approach as described in [Joseph 95].

Object classes implementations on different components are responsible for logging operations and applying them in the desired order. As suggested before, each server will apply all updates to the local copy of the object. The class programmers should select the ordering constraints for updates application that conforms to the expected objects' semantics.

In order to allow flexibility, each object is composed of several components (subordinated objects): a *capsule object* that is responsible for the aggregation of all components that compose a class and implements the interface to the base system; an *attributes object*, that is used to maintain relevant meta-information; a *log object* that deals with operations logging; a log *ordering object*, that orders updates' application in a type-specific way; and *a data object*, that represents the real data type with its own state and operations. Our system provides standard implementations of the different components of the object class but the data object. This is the only component that the programmer must provide if the ordering, attributes and logging semantics already available fits her or his objectives.

When some desired component semantics is not available, a new implementation or an extension of an existing one should be built. We currently have a default implementation for log, attributes and capsule objects and several ordering objects: no-order, causal order, total order based on a primary replica scheme (to enable fast update commit as suggested in [Terry 95]) and optimistic total order adopting an undo/redo scheme. This support implements an optimistic total ordered object that uses operations semantic information to deal with concurrent updates and that would notify users when it is unable to automatically resolve conflicts[5].

Using the JavaBeans component model and our persistent and replicated object-storage model, we can easily implement the component required to act as an event adapter filtering logged updates managed by the object-storage infrastructure.

The generic model for the event-notifier is represented in the following figure:

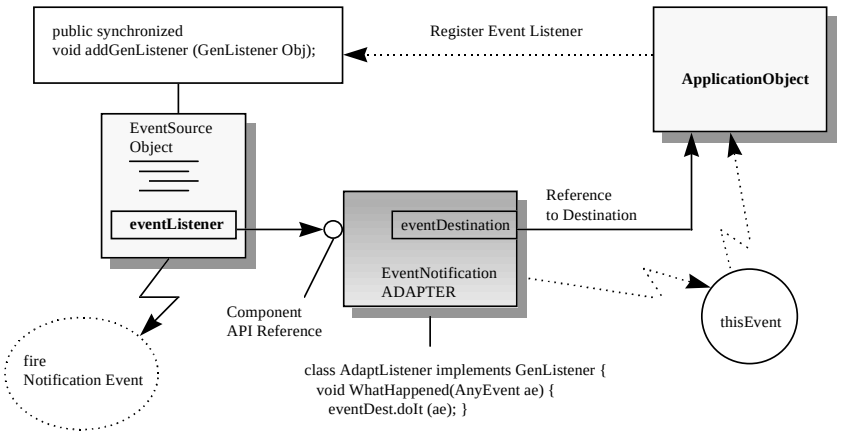


Fig. 5 The event adaptor generic model

5 In the base support, we intend to develop other possibilities, such as for example a capsule object that allows for the support of a stable and a tentative data object, as proposed in [Terry 95].

The primary role of the generic event-adapter component is to conform to the specific event listener interface (event consumer) expected by the event source (event-notifier), and to decouple each incoming event notifications on the input interface from each actual listener. From the applicational viewpoint the event adapter is intended to be used as a tailorable filter.

As the object-storage infrastructure provides different potential event sources notifying different classes of events concerned with the data-objects managed, the most useful approach to generalize the event filtering technique is to build a generic event demultiplexer. Any specific event listener object may only implement one instance of a specific event listener interface. Thus, if each listener registers itself with multiple event notification source objects for the same class of events in the base storage support, the listener has to determine for itself, which source actually emitted each specific event it wants to be notified. The notification service must provide the following functionality: different event sources will be delivered to different target methods on the final listener object. For this purpose, a demultiplexing adaptor (DMUX) must be interposed between each source and the listener. Each DMUX takes an incoming method call of one session name and then calls a differently named method on its target object.

One can imagine, for example, that in the context of each application, we can manage two buttons on a *DialogListeningBox* object: OK and CANCEL, both of which fires a *buttonPush (event)* method. The *DialogListeningBox* is designed to invoke the methods: *doOKaction()* for the button OK and *doCancelAction()* for the Cancel button. So the *DialogListeningBox* defines two classes: OkButton() and CancelButton() which are adaptors implementing a Listener interface, but dispatching the notifications to specific action methods.

The implementation is represented in the fig.6, when the component adaptor is used in a final component-tool .

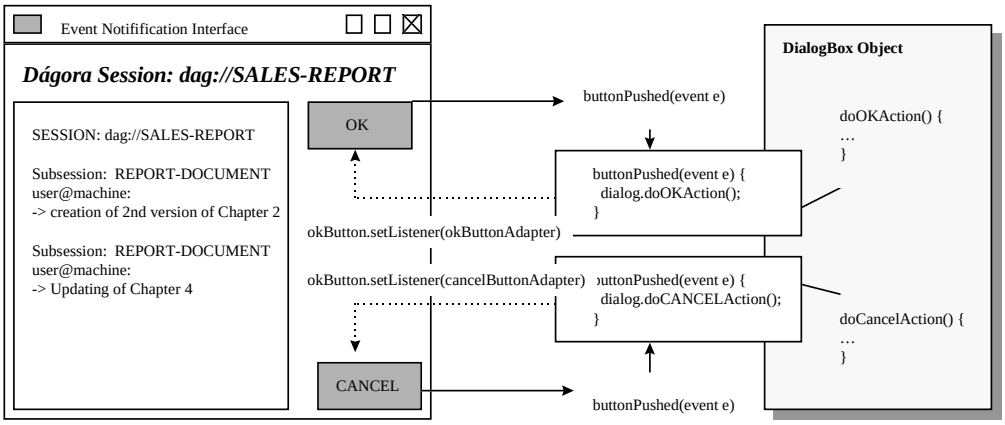


Fig.6 The materialization of the EventNotification component

The simple DMUX model described has yet a drawback: the number of private adaptor classes can become very high because of the large number of different event targets, as a different adaptor class is required for each event source occurring in a global session. Remember that several tools can be used in the context of the same session. When there are a large number of event sources, another technique can be employed in order to obviate the need of a new adaptor per source target. The idea is to extend the basic model above in a way that generic adaptors can be automatically generated by specific application builders. This involves synthesizing each adaptor class that uses the new low-level Java introspection APIs to map new classes of events originated in multiple source instances onto specific method invocations. In fact, JAVA provides a basic support for this approach, via the invocation of a special method *(invoke)* of a native class called *java.lang.reflect.Method.*

The technique allows the introduction of a generic multiplexing adaptor as an alternative for a proliferation of simple multiplexing adaptors, as illustrated in the fig. 7. However, the manual use of that technique is not recommended because of the complexity involved. Moreover, it introduces possible fatal errors in applications using that code at runtime. So, it is strongly recommended that this technique should be supported by an application builder, where great constraints may be applied to, and subsquently checked during the code generation, reducing the risk of introducing fatal runtime errors into the application code.

JavaBeans provides this kind of facilities in the form of a general EventListener interface, that is the prototypical interface from which all event listener interfaces shall be derived. This interface defines no methods, and only exists as

a way of identifying sub-types as event-listeners. By convention, all derived EventListener interfaces shall specify the event notification methods, according to a recognizable design pattern, which is defined by a standardized method. Adopting these definitions, a JavaBeans application builder can be adopted to facilitate the support of the component model of the EventAdapter.

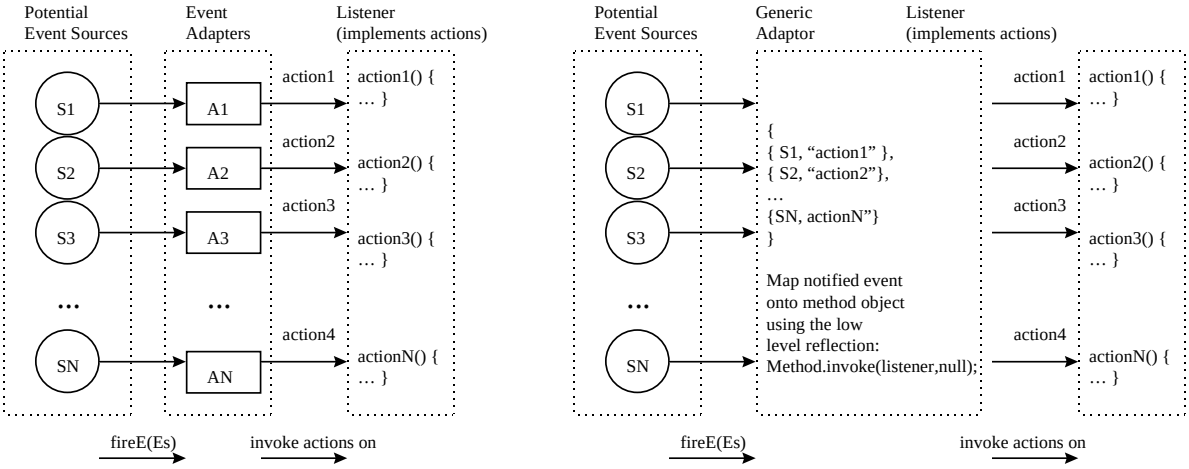


Fig.7 Proliferation of adaptors vs. a generic multiplexing adaptor component

Assuming that in the near future the run-time support for java objects tend to be generalized in different software products, the potential of this approach for system integration is considerable. We also expect a close cooperation between collaborative software components produced this way and other components produced with environments like activeX or CORBA, even if we are aware of the current on-going development of the JavaBeans technology.

7. Outlook and summary

This paper discusses several viewpoints about tailorability issues in groupware. We identify several directions of research to find solutions and methods to produce tailorable groupware. We have also illustrated how these classes of solutions fit in the context of a scalable generic distributed system platform and framework for large scale CSCW. Different directions for the tailorability support are focused on different viewpoints and emphasis according with the different support levels involved in that framework.

The base supports recognized for this framework are:

- group communication and replication services, transparently used by means of well-defined low-level APIs re-used in the context of applications and its functional components;
- group memory (group collaboration persistence) supported by a specific replicated object-storage, which is a fundamental service to achieve the scalability criteria in large scale;
- A well-structured component-oriented design, which involves:
  - The definition, implementation and validation of useful components **(what common components)**
  - The selection of which components must be tailored in a generic way **(which tailorable components)**
  - The design of such components in the appropriate levels of the framework **(how to structure components)**
  - The adoption of specific programming support for integration, composition and implementation of components which requires the adoption of a component-based architectural model **(how components fit together)**

The paper also proposes a generic and configurable coordination model based on common coordination components and basic tailorable mechanisms supported by the notion of *collaborative session*. With this support, specific applications used in each session concentrates on task-specific collaborative functions and in the modeling of specific workgroup methodologies. It is important to emphasize that this approach do not means that the development of generic distributed system platforms for collaboration must allow a standard model of cooperation/coordination and organization. On contrary, these systems must allow a diversitry of coordination models to work in a complementary and adjustable way.

In the final part of the paper, we illustrate the development of specific components by adopting a component-oriented architectural model. In the case we use JavaBeans, which in the framework can act as a software component model reusing the fundamental supports of the framework (which are written in JAVA).

In summary, the two main directions for the support of tailorability in CSCW that we have identified are:

The Coordination and Management perspective. This direction deals with customization, tools integration and workspace configuration issues supported by means of a coordination model and its support mechanisms at computational level.

The Software (Groupware) Engineering perspective. This direction deals with the transparent adoption of base supports provided in a generic framework and with the need of flexible support for collaborative-oriented adaptable and reusable software components. These groupware components must be used as specific building blocks in the context of an open integrated architecture.

Bibliography

[Knister92] Knister, M., Prakash, A., "DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors", Proceedings of the CSCW'90, Oct 1990.

[Roseman92] Roseman, M. and Greenberg, S., "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications", Proc. ACM CSCW 92, Nov 1992.

[Cosquer96] Cosquer, F. J., "Large Scale Distribution Support for Cooperative Applications", Phd Thesis, Univ. Técnica de Lisboa, Mar 1996.

[Kindberg96] Kindberg,Tim., Coulouris, G., Dollimore, J. and Heikkinen, J., "Sharing Objects over the Internet: the Mushroom Approach", proc. Of Global Internet 96, IEEE, London, November, 1996

[Dom97a] Domingos Henrique J., Martins J. Legatheaux, Simão, J. , "A Flexible and Integrated Middleware Framework for the support of Larg Scale Cooperative Work", in proc. of the international conference on system sciences, vol. 1, pp 82-92 Hawai, Jan 1997

[Batteram97] Batteram, H., Idzenga, H., "A Generic Software Component Framework for Distributed Communication Architectures", ECSCW'97 Object Oriented Groupware Platforms Workshop, pp 68-73, September, 1997.

[Renesse94] Van Renesse, R.t, Takako, M. Hickey and Birman, K. P., "Design and Performance of Horus: A Lightweight Group Communication System", TR 94-1442, Cornell University, Aug 1994.

[Simão97] Jorge Simão, N.M.Preguiça, Henrique J.L., Domingos, J.A. Legatheaux Martins, "DÁgora: A Flexible, Scalable and Reliable Object-Oriented Groupware Platform", presented in the ECSCW97 OOGP - Workshop on Object-Oriented Groupware Platforms, September 1997, Lancaster, England.

[Dom97b] Domingos H J., Martins J. A.L, Simão, J.M. , "Support for Coordination and flexible synchronicity in large scale CSCW", in proc. of the CYTED-RITOS - CRIWG´97 - 3rd International Workshop on Groupware, pp 81-90, Madrid, Spain, October 1997

[Kistler91] Kistler, J., Satyanarayanan, M., "Disconnected Operation in the Coda File System", in proceedings of 13tth ACM SOSP, 1991

[Joseph95] Joseph, A. D:, et al., "Rover: A Toolkit for Mobile Information Access", in Proc. Fifteenth Symposium on Operating Systems Principles, (SOSP) December 1995

[Terry95] Terry, Douglas B., et al., "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System", Proc. 15th Symposium on Operating Systems Principles, (SOSP), Dec 1995

[JavaBeans97] Ref. "http://java.sun.com/beans" for the current specifications and programming supports provided for the JavaBeans components model.

[JavaSoft97] JavaSoft Inc., "The JavaBeans 1.0 API Specification", Version 1.00-A, December 1996.

[Birman 94] Birman, K.P., and Renesse, R.V., "Reliable Distributed Computing with the Isis Toolkit", IEEE Computer Society Press, 1994.

[Renesse 94] Van Renesse, R.t, Takako, M. Hickey and Birman, K. P., "Design and Performance of Horus: A Lightweight Group Communication System", TR 94-1442, Cornell University, Aug 1994.

[Cheriton 85] Cheriton,D., Zwaenepoel, W., "Distributed Process Groups in the V Kernel"- ACM Transactions on Computer Systems, Vol.3, n.2 pp77-107, 1985.

[Kaashoek 91] Kaashoek, F. amd Tanenbaum, A., "Group Communication in the Amoeba Distributed Operating System", IEEE Proc. International Conference on Distributed Computing Systems, pp.-230, 1991.

[Amir 92] Amir, Y., Dolev, D., Kramer,S., and Malki, D., "Transis: A Communication Sub-System for High Availability", porc. 22nd International Symposium on Fault-Tolerant Computing - IEEE, July1992.