Communication Framework for XMSF

Dr. Norbert Schiffner
Secure Distributed Technologies
Fraunhofer CRCG
Providence, RI 02903
Email: Schiffner@crcg.edu
Phone: (401) 453 6363 x103

ABSTRACT

The steadily increasing quality of hard- and software enables the development of more and more sophisticated groupware. However, modern distributed systems challenge the abilities of groupware manufacturers, due to the complexity involved in the development of those systems. Frameworks for collaborative environments might provide a solution to these problems, helping to reduce complexity in groupware development by provision of suitable software components.

The article then introduces the HOUCOM framework for collaborative environments.

INTRODUCTION

Collaborative environments provide human users with the mechanisms and means they need in order to perform their collaborative tasks within dislocated workgroups, i.e. those groups whose members are spatially separated from each other. The inter-member collaboration is implemented by means of groupware, combinations of soft- and hardware that efficiently support group members in their collaborative tasks.

But groupware usually faces many challenges and problems which must be efficiently accepted and solved by the developer of groupware systems. These problems are not only of technical nature but also take the “human factor" into account. Whereas a variety of hardware components for groupware exists, the software that embeds those hardware and adds value to them is still underdeveloped and usually falls behind the demands on modern collaborative environments as imposed by the steadily growing user community.

Challenges of Distributed Systems

Distributed systems are characterized by their ability to share media across a network among two or more workstations, allowing more than one person at the same time to generate process, represent and store information. Some of the most important challenges of distributed systems which most of them are also valid for the XMSF framework are discussed in the following paragraphs.

Consistency of distributed data. A distributed system replicates its data on the hosts that are remotely connected to that system. To make sure that each such remote host has the same view onto the distributed media, the sending host has to employ mechanisms that guarantee the consistency of the data being distributed.

Especially in transaction-based systems it is essential to preserve the principles of atomicity (either everybody or nobody receives the message), first-in-first-out ordering (messages arrive in the order they were sent) and causality preservation (no message B indirectly overtakes a message A that has been sent earlier, and on which message B is causally depending, i.e. no B without A).

Efficient session management. Session management includes floor control (access control to a session) and session control (activity control inside a session). Distributed systems have to offer appropriate mechanisms that enable an efficient conference management in different conference layouts ranging from one-to-one over one-to-many up to many-to-many communication structures. Certainly, this session management has to assure a consistent view onto the conference's state on all participating hosts.

Network as a “black box”. Distributed systems distribute their data via a network and usually do not much take care of the network's state, enabling a transparent end-to-end communication from host to host. However, performance, reliability and security issues force distributed systems to embed mechanisms for overcoming the problems inducted by this “black box approach”: the existence of real-time restrictions and synchronization in these systems requires the consideration of the network performance, including latency and throughput which is partially covered by the Resource Reservation Protocol (RSVP) and the Real-time Control Protocol (RTCP) which is integral part of the Real-time Protocol (RTP); the requirement for fault-tolerance imposed by certain media can either be deferred to the network (by using reliable transport protocols such as TCP/IP), or implemented by the system's components (e.g. error-correction codes embedded in the data stream or “sliding-window” protocols); the demand for data protection against improper use implies an additional encryption mechanism on the data representation level.

Because of their distributed nature, distributed systems usually involve different people working with different hard- and software. This heterogeneity is subject of the following paragraphs.

Heterogeneity of users. An arbitrary number of human users with all their weaknesses and strengths meet each other in a conference / session. Each of these may have different demands on the system they use, and on the media they share. People with eye defects for instance usually need a post-processing of the incoming media streams (e.g. a contrast enlargement as in the case of video streams), whereas healthy people do not.

Heterogeneity of hardware. Potentially each host participating in a conference can have a hardware configuration that is different from each other participating host. Different hardware implies different levels of performance: one machine may have installed a powerful video encoder on it, whereas another machine within the same conference may have installed no such decoder at all; in the latter case, the system must be able to emulate such a decoder by software which certainly is not as performant as a hardware solution.

Heterogeneity of software. The same as with hardware is valid for software. This includes different operating systems and drivers on different hosts. However, a distributed system must still be able to interconnect these people regardless of their (considerable) different software configurations.

Heterogeneity of communication. Different media employ different encoding and decoding standards (aka. “codecs") that transform the raw data stream into a more compact data representation and vice versa. Video streams for instance are usually using the MPEG7 or H.2618 standards for compression. A host that wants to view this coded video stream must have decoder implementations of these standards installed either on hardware or emulated by software. Otherwise, he would not be able to decode and view those video streams. For a conference, this implies that the participants must agree on a common codec for each media type to be transmitted.

This also holds true for communications protocols set on top of data, such as the Real-time Protocol (RTP) or the Real-time Streaming Protocol (RTSP).

The User's Demands on Modern Groupware

The previous subsections have identified the technical demands distributed systems. The system agent, i.e. the software a human user employs in order to participate in a conference, is a critical piece of a distributed system. An important issue is the general acceptance of the groupware implementation. This subsection is going to identify the demands that user might impose on their individual groupware solution. All these demands are considered to be essential prerequisites for achieving general acceptance in the groupware user community.

Interoperability. The participants within a conference may use different system agents being based on different hard- and software. This heterogeneity must be managed by groupware in order to allow all users to work together anyway. This usually implies the use of commonly accepted communication standards, including networking and media coding standards. However, those standards alone cannot guarantee the interoperability of groupware solutions. Rather, the groupware must be able to dynamically switch to those standards as this becomes necessary.

Interoperability also refers to the ability of groupware to enable collaboration between those users that employ groupware applications of different vendors.

Flexibility and extensibility. The environmental conditions under which a system agent runs may change over time. This might happen because the user aims for altered goals, his preferences have changed or his system has to be updated for any reason. Groupware has to become enabled for those evolutionary changes, a demand that monolithic system architectures generally do not meet. Flexibility refers to the property of a groupware to exchange functional components by another comparable component, whereas extensibility addresses the ability to embed additional functional components into the groupware. Fulfilling these two principles enormously increases the groupware's expected lifetime.

Scalability. One and the same groupware implementation can be used in different scenarios, involving different demands on the system's performance. One scenario includes a simple one-to-one conference, whereas another scenario might include a complex many-to-many conference. Therefore, groupware has to be scalable in order to support those varying environmental conditions. This primarily concerns the session management and the data transport mechanisms as well.

Ergonomy. Under certain circumstances, it is desirable to be able to access infrastructures that are provided by a network, but that are not an elementary part of the collaborative environment itself. Good examples are the RealVideo and RealAudio (www.realnetworks.com) servers being widely available on the Internet. Several broadcasting stations, including CNN (www.cnn.com) and ABC (www.abcnews.com), provide free transmissions of their newscasts over those servers. A groupware that allows the user to connect to those servers adds value to itself, thus helping to improve the groupware's general acceptance.

Usability. A user expects his employed groupware application to be usable; this means that the software must efficiently support him in his collaborative tasks, and that the operation of the system is kept as simple as possible. Too much (unneeded) functionality and information overload work against the principle of usability.

Thus, customizability of the user interface and the software's modular architecture are the keys to success, because they allow the user to decide on what functionality to embed into the groupware. This emphasizes and conforms the demand for extensibility and flexibility as explained above.

Frameworks: A Technical Solution?

All Subsections above indicated a high complexity a developer of groupware usually has to deal with. Decisions this developer has to make include for instance decisions on the supported conference management strategies and the standards to be implemented into the groupware. Once these decisions have been made, the developer now has to realize them, which commonly is much more difficult, especially when related previous experience is not available.

Managing the complexity in groupware development is an essential task since it decides on whether the project succeeds or fails. Managing complexity in this context means that the developer must be efficiently supported in several phases of his development cycle, including the design, the requirement specification, implementation and test phases.

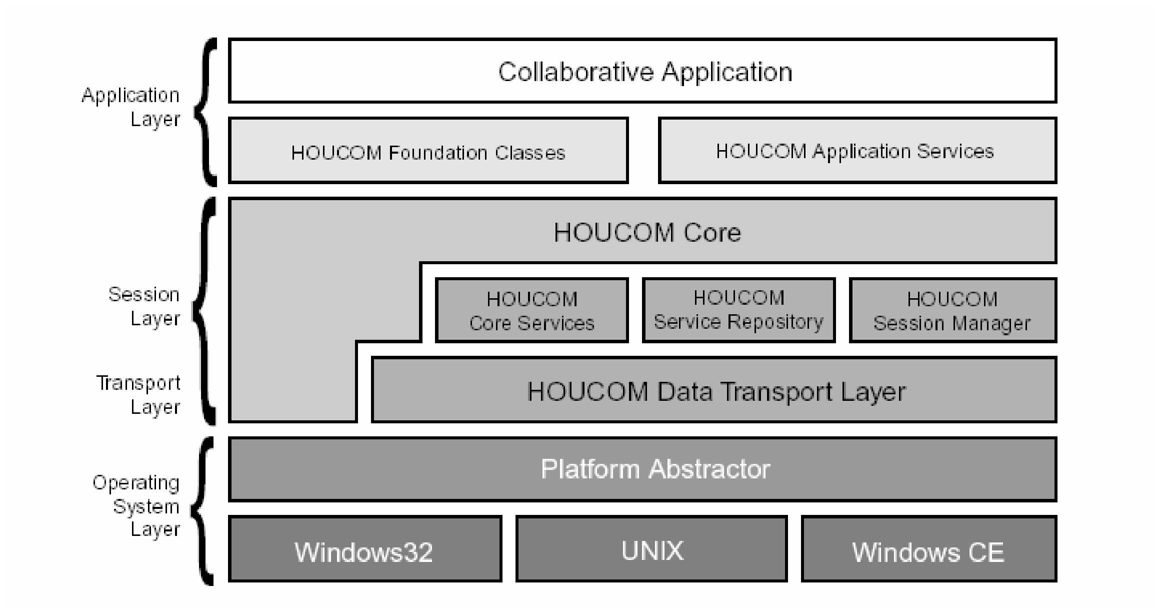
Frameworks that support the development of groupware must be able to support the developer in those tasks. A framework usually consists of reusable classes, modules/libraries and executable components that aim for a common goal, here: the implementation of a collaborative environment. Developers use these frameworks in order to simplify their design and implementation phases by falling back on parts of the framework. Thus, making use of frameworks simplifies design and implementation of groupware; this in turn accelerates the development process, leading to shorter projects, lower development costs and hence lower product prices still the most important buy criterion for potential customers. Further, frameworks develop over time and integrate the experience of many people and many previous projects; therefore, the developer can expect those frameworks to be relatively robust, reliable and efficiency in contrast to self-developments that generally lack of those guarantees due to their early development state. In short, the employme nt of frameworks enable shorter projects resulting into higher-quality products.

A framework for collaborative environments may help developers of groupware by

- providing robust tested and optimized components,
- implementing complex strategies, e.g. for conference management, and (pseudo-) standard protocols, e.g. the Session Announcement Protocol (SAP),
- providing generic interfaces which transparently hide underlying complexity from the caller, e.g. network interfaces that abstract access to network,
- providing executables for special purposes, e.g. session managers, LDAP servers or servers; at this point, frameworks are advancing to integrated architectures.

HOUCOM Architectural Overview

HOUCOM provides the developer of distributed cooperative systems with a set of C++ classes and run-time modules that enable him/her to develop those systems much more efficient and reliable.



High-level View on HOUCOM

HOUCOM consists of a set of high-level elements:

- The HOUCOM Core is the basic module, that is responsible for the most fundamental collaborative functionality (e.g. session and conference management) and access to additional, more sophisticated functions, which are required in distributed and cooperative systems in particular.
- The HOUCOM Service Framework enables the modularization of component-based systems by means of reusable software modules (aka. "components"). It allows the deployment of commonly required components in specialized servers ("Service Repositories"), from which a client can download and install the necessary components on the localhost.
- The HOUCOM Data Transport Layer provides applications with a unified transport layer, that transparently hides the complexity usually involved with network programming and communication protocols. This layer supports many transport configurations and styles, including TCP/UDP unicast, broadcast and multicast. This allows an application (servers in particular) to decide on its data transport configuration at run-time, depending on the network capabilities and constitution.

Besides these elements, a number of applications are supporting the HOUCOM framework in accomplishing its tasks:

- The HOUCOM Session Manager is a specialized server, that runs on a host and that can be accessed by each conference participant individually. This server is responsible for the provision of a reliable session management, including floor and session control. Future versions of HOUCOM will replace the centralized Session Manager through a more flexible distributed session management architecture, in which a set of those Session Managers is available to maintain more complex conferences. The Session Manager is entirely controlled though the HOUCOM Core System and usually not visible to the client applications.

- The HOUCOM Service Repository is a server being specialized on the storage, maintenance and supply of HOUCOM Service Components in the context of the HOUCOM Service Framework. Clients, that require a specific component, can browse the available Service Repositories and download the required component from one of these repositories. This approach makes components in HOUCOM ubiquitous, thus allowing for a very high interoperability between different users on different platforms (assumed that components are available for all target platforms).

The subsequent sections describe these elements of HOUCOM in detail.

The Core System

Each HOUCOM-enabled client application is linked with a Dynamic-Link Library (DLL), that encapsulates the HOUCOM Core System. This module provides basic support for distributed systems as well as the event handling on which HOUCOM is based on.

The Core provides the programmer with a rich set of C++ classes, which hide the complexity of distributed systems. As HOUCOM Core evolves in future, updates of the interface may become necessary. The Core has been designed in a way, that allows the support of different interface versions, which is essential for keeping the Core backward-compatible, thus avoiding it to make changes to existing client source code.

The Service Framework

Commonly required functionality in HOUCOM is encapsulated in small functional code modules aka. "HOUCOM components". Each component provides its functionality to a client through one or more C++ interfaces. Those components can be linked to a client application at run-time (run-time linking instead of load-time linking!), thus allowing an application to flexibly decide on what components to use depending on the current situation.

The Service Framework is not based on popular component middleware, such as (D)COM or CORBA. Instead, a new middleware has been developed for several reasons:

- Platform independency: the complete Service Framework can be easily ported to all operating systems, that support dynamic linking and reliable network communication.
- Usability: those components, that are required by a client application, but that are not stored on the localhost, can be automatically downloaded from one or more nearby HOUCOM Service Repositories.
- Performance: because HOUCOM components are mapped into a client's process space, access to a component is without any overhead. This enables components to encapsulate even the most time-critical code.
- Efficiency: the framework can transfer and handle components in a compressed format that reduces size of components by 60% in average, thus leading to much

shorter transmission times between clients and service repositories and less space consumption on storage devices (important for low-resource devices, such as handheld devices).

- Commercialization: future versions of the HOUCOM Service Framework will support commercial licensing of components, using different license types, including "time-restricted licenses" and "functionality-restricted licenses".
- Security: future versions will add advanced security mechanisms to components in order to protect clients form loading and activating malicious components (e.g. components modified by viruses).

The Data Transport Layer

The Data Transport Layer (DTL) is HOUCOM's standard mechanism for exchanging data with remote applications by means of inter-process communication, such as network sockets, pipes, or shared memory. A client can use this layer in order to receive data from or write data to a communication partner.

The DTL introduces "Data Transport Objects" (DTOs), which contain a directed acyclic graph of interconnected Data Processing Units (DPUs), that transform incoming data from one representation into another representation. This concept allows for a very flexible configuration of communication services.

The topology of this graph is determined either by the client, or by a XML-based profile, that can be downloaded from a profile database. For instance, two programs, that want to communicate, could share the same profile in order to setup an appropriate communication channel, which facilitates a consistent and compatible data exchange.

Data Processing Units can be provided either by a client or by HOUCOM components.