Proceedings of the 3rd International Workshop on
Protocols for Multimedia Systems (PROMS'96)
October 1996, Madrid

Design of a shared whiteboard component for multimedia conferencing

Marten van Sinderen, Phil Chimento, Luís Ferreira Pires
<*sinderen,chimento,pires@cs.utwente.nl*>
Centre for Telematics and Information Technology
University of Twente

Abstract

This paper reports on the development of a framework for multimedia applications in the domain of tele-education. The paper focuses on the protocol design of a specific component of the framework, namely a shared whiteboard application. The relationship of this component with other components of the framework is also discussed. A salient feature of the framework is that it uses an advanced ATM-based network service. The design of the shared whiteboard component is considered representative for the design as a whole, and is used to illustrate how a flexible protocol architecture utilizing innovative network functions and satisfying demanding user requirements can be developed.

1. Introduction

The Platinum project[1] aimed at developing distributed multimedia applications, supported by broadband network technology. This project takes account of both user and technology perspectives in order to enable sophisticated services that support the actual end-user needs. The concrete objectives of the project are:

- design, implementation and analysis of an innovative, ATM-based, broadband *network platform* that supports multimedia multiparty communication services;

[1] The Platinum project has been partially supported by the Dutch Ministry of Economic Affairs. Partners in the project were: Lucent Technologies (formerly AT&T), Telematics Research Centre, and Centre for Telematics and Information Technology. The project started in October 1994 and ended in June 1996.

- design and implementation of a *middleware platform* that sits on top of the network and supports the generic functions of distributed multimedia applications;
- design and implementation of a *multimedia conferencing application* that uses the middleware and that itself can be used in a tele-education context.

This paper presents the overall architecture of the Platinum design and further focuses on the protocol design of a shared whiteboard application, which is part of the multimedia conferencing application. The shared whiteboard application allows users to manipulate a set of information objects, while maintaining a common view of this set of objects. As such, it can be used as a building block in more specific applications, such as collaborative editing. The shared whiteboard application offers a special shared whiteboard medium type to its users, which is one of the medium types supported by the multimedia conferencing application. Any multimedia conferencing user can participate in zero, one or more media, including the shared whiteboard medium, in which case the user is (as well) a shared whiteboard user. The set of shared whiteboard users can be dynamically changed, requiring interactions between the shared whiteboard application and the control part of the multimedia conferencing application.

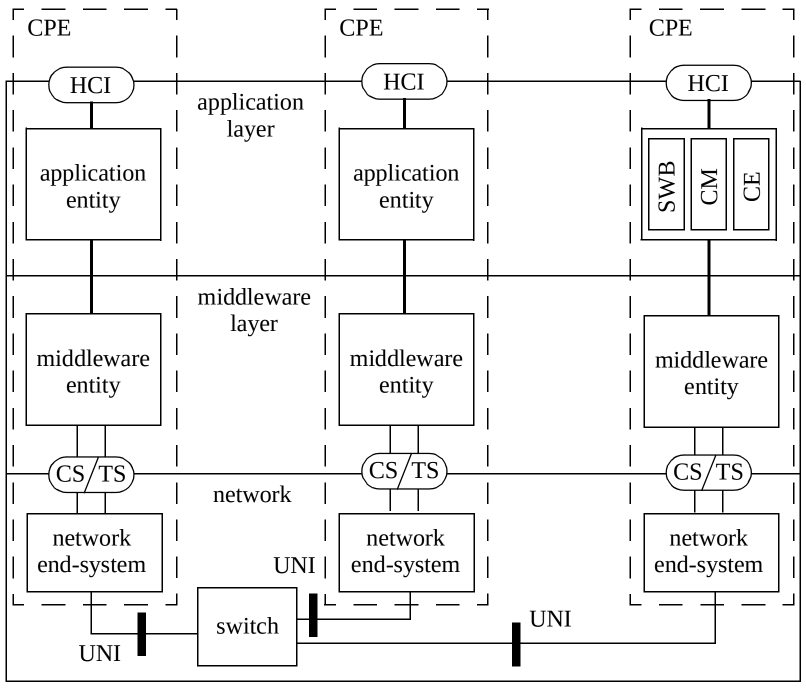
Many multimedia conferencing systems, some with a shared whiteboard facility, have been developed in the past (e.g., [3, 7, 11]). In essence, the Platinum multimedia conferencing application does not offer more, or more advanced, functions compared to these systems. Its design is interesting, however, for two reasons. First, it serves as a test environment for the network platform, which does embody a number of innovative functions; the Platinum application had to exploit, via the middleware, these functions. Second, the Platinum application was designed with a strong requirement for flexibility: it should be easy to extend and modify, and so it provides a sound basis for developing other applications. The flexibility requirement motivated the decomposition of the Platinum application into a middleware layer and an application layer: the former should provide a platform for developing distributed multimedia applications in general, the latter should be used as a basis for developing applications in the tele-learning application domain. This paper therefore highlights the design approach and the architectural decisions, in particular related to protocols, that underlie the Platinum design. Details of the architecture are only discussed with respect to the shared whiteboard application, which is considered representative for the Platinum design as a whole.

The objective of the paper is to show how a flexible protocol architecture utilizing innovative network functions and satisfying demanding user requirements can be developed. The paper also gives an outlook of further work that will be done in follow-up projects.

The remaining of this paper is organized as follows: Section 2 outlines the overall architecture of the Platinum design; Section 3 presents the design and the resulting architecture of the shared whiteboard application; Section 4 discusses some possible future extensions and alternatives related to the shared whiteboard design; and Section 5 introduces some relevant discussion points related to the Platinum design.

2. Overall architecture

Figure 1 depicts the overall architecture of the Platinum design. The architecture consists of a layered structure, which is further explained below, as well as a collection of distributed end-systems (or Customer Premises Equipment). Only three end-systems are shown in Figure 1; the elaborated part in the rightmost end-system represents an internal structure in terms of application components, which are explained below.



CPE = Customer Premises Equipment	CE = Collaborative Editing
HCI = Human Computer Interface	CS = Connection Server API
SWB = Shared Whiteboard	TS = Telecommunication Service API
CM = Conference Management	UNI = User Network Interface

Fig. 1. Overall architecture of the Platinum design

2.1 Tele-communication services

The network platform built in the Platinum project supports calls between multiple users, where each call consists of a set of connections. Each connection is capable of supporting some specific medium type and involves any non-empty subset of call users. The network platform therefore supports multimedia and multiparty communication services. The current implementation only employs a single ATM switch (with co-processor) which is connected to all network end-systems. The interworking between the ATM switch and each network end-system is implemented by a User Network Interface (UNI) protocol stack consisting of Q.2931, AAL, ATM and Physical layer signalling protocols. The users of the network platform (i.e., the middleware entities) interact with the network (i.e., the network end-systems) via a Telecommunication Service (TS) Application Programming Interface (API). ATM is claimed to be a best match for multimedia application requirements (e.g., see [9]); one of the goals of the Platinum project, which is not further discussed in this paper, is to investigate this claim.

2.2 Middleware design

The middleware layer consists of middleware entities on which multimedia applications can be developed. It provides a cooperation context, called *session*, to its users (the application entities) which can be easily extended and modified to suit new or special application requirements. Two aspects of this layer have been distinguished in the project: the protocol architecture, which is concerned with the proper interworking of middleware entities in different end-systems, and the software architecture, which implements the protocols and local support functions in a highly modular fashion, in order to satisfy the flexibility ('easy to extend and modify') requirement.

The *protocol architecture* identifies protocol building blocks, such as start_session, close_session, add_to_session, delete_from_session, modify_session and medium_ data_transfer, which together provide a rich conferencing service. The building blocks either manage the use of the network service via the TS-API, add additional control by exchanging control information as data on the (user- or control-plane) network connections, or accomplish the transfer of medium data across user-plane network connections.

The *software architecture* uses object-oriented technology, motivated by the flexibility requirement mentioned above. Its top level structure is based on the Model/View/Controller (MVC) design pattern (e.g., see [2]), which supports a separation of the following middleware aspects: maintenance of dynamic context information about a session ('model'), use of the session ('views') and management of the session ('control'). The identification of classes, their responsibilities and their relationships is based on the application of the MVC pattern. 'Control' classes are mostly responsible for the interaction with the network through the TS-API (control-plane); 'view' classes are mostly related to the use of network connections (user-plane).

2.3 Applications

In order to determine useful applications of the middleware platform, user requirements from a tele-education context were selected and analysed. The *tele-education context* is formed by educational activities at the University of Twente, in collaboration with other technical universities. As a result of this activity, three applications were identified: (multimedia) conference management, shared whiteboard manipulation and collaborative editing.

The application-specific parts of these applications form the Platinum application layer, while generic aspects are supported by the middleware layer. The *conference management* application, with functions for establishing and releasing associations between participants and media, and for controlling the concurrent use of media, provides a conferencing framework in which the latter two applications can be used. The *shared whiteboard* application can be seen as the provider of a special medium type: a collection of textual and graphical objects, which can be manipulated by each of the participants associated with this medium. It is the responsibility of this application to ensure that the participants have a consistent (shared) view of the objects, and that newly associated participants are informed of the current view. The *collaborative editing* application supports joint editing of document parts which together form a compound document. Also the compound document can be seen as a special medium type, structured in terms of document parts, whose view among the associated participants must be kept consistent by the collaborative editing application. As with the middleware layer, the architecture of the application layer has a protocol and a software perspective.

The overall architecture depicted in Figure 1 is a static model of the main responsibilities identified in the Platinum design, common to both a protocol and a software perspective. A protocol architecture is a further elaboration of the overall architecture: it captures interoperability requirements, by identifying and structuring protocol functions. Also a software architecture is a further elaboration of the overall architecture: it captures requirements on software components by identifying and structuring (relating) object classes. Since the software architecture must consider both local and interoperability (interworking) aspects, the protocol architecture imposes functional requirements to the software architecture.

A prototype that partly implements the Platinum application and middleware software architectures has been demonstrated during the project closing meeting in June 1996.

3. Shared whiteboard architecture

This section presents an overview of the protocol architecture of the shared whiteboard application and briefly considers the software architecture and the interactions with the control part of the conferencing application.

3.1 User requirements and service perspectives

Starting point for the development of the shared whiteboard protocol and software architecture was a set of user requirements which followed from the analysis of a tele-education context. The main requirements are:

- *viewing* and *manipulation* (addition, modification and deletion) of *shared* information objects, including text and graphical objects; and
- assignment and de-assignment of an *exclusive right* to manipulate one or more information objects.

The following key design decisions are derived from these requirements:

- a user *views* information objects through a local copy;
- to provide the illusion of *shared* information objects, local copies are only changed under control of the shared whiteboard application layer. The shared whiteboard application layer must ensure that the same manipulations are applied in the same order to all local copies. These manipulations do not have to be performed necessarily at the same time, since possible temporary inconsistencies are not noticeable by the users;

- the control exercised by the shared application layer implies that each user must submit its *manipulation* requests (operations) to the shared whiteboard application layer. The shared whiteboard application layer accepts or refuses operations: accepted operations are forwarded to all users, which receive them in the same order, whereas refused operations are only indicated to the requesting users;
- users can obtain the *exclusive right* to manipulate certain information objects by submitting a select operation; they can give up a previously obtained right by submitting an unselect operation.

The model of the shared whiteboard application which results from these design decisions is shown is Figure 2. The model identifies local applications and a control part, and interactions between local applications and the control part. The interactions represent operation requests and indications of accepted and refused operations. The control part represents the shared whiteboard application layer, whose (distributed) implementation is not yet considered.

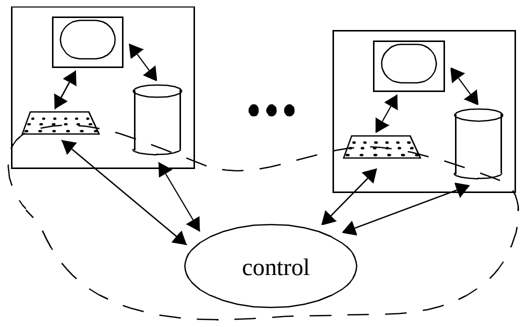


Fig. 2. Model of the shared whiteboard application

The integrated view of the control part and the interactions with its environment (local applications), can best be described with a service definition. We consider here two aspects of the service definition: an object-oriented view and a user-oriented view. Figure 3 describes these views by means of state transition diagrams.

The object-oriented view considers the different interactions and their relationships which apply to a single information object. The user-oriented view considers the different interactions and their relationships which involve a single local application (i.e., a service user). The user-oriented view reveals that a distinction is made between a confirm and an indication interaction: a confirm interaction informs a user about the outcome (accepted or refused) of a previously requested operation, and an indication interaction informs a user about an accepted operation requested by some other user.

3.2 Protocol layers

The protocol architecture of the shared whiteboard application serves to identify and structure the protocols that implement the service views mentioned above. The following additional design decisions form the basis for the protocol architecture:

- each local application is locally supported by a *user agent* in the application layer;

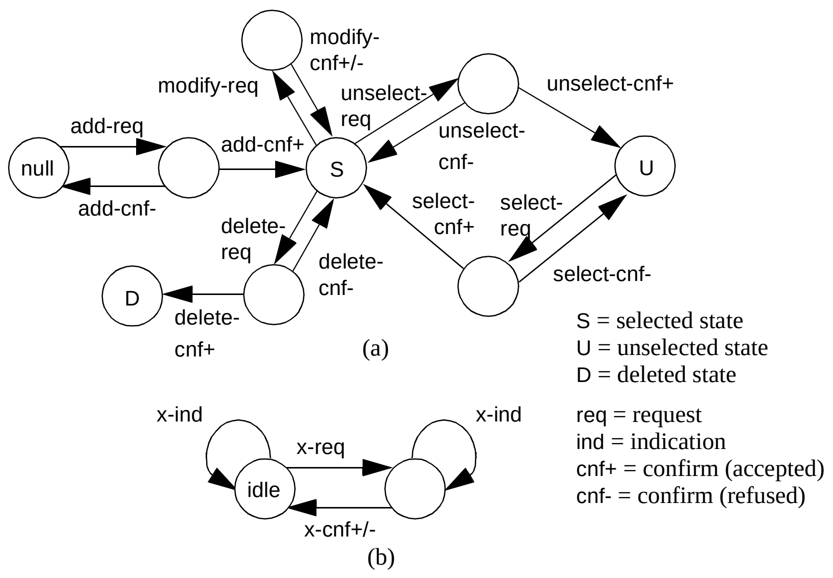


Fig. 3. (a) Object-oriented service view and (b) user-oriented service view

- user agents communicate with a central *controller* in the application layer. The central controller resides on an end-system whose participation in the shared whiteboard application is committed. A local application with committed participation is termed a critical user for this reason;
- the central controller receives operations from the user agents. It decides which operations can be accepted and which must be refused, broadcasting a positive indication to all user agents for each accepted operation and returning a negative response to the requesting user agent for each refused operation.

Provided that messages exchanged between the central controller and a user agent are not lost and are not re-ordered, accepted operations can always be passed to the local applications in some unique order. Multicast with both guaranteed delivery *and* order preservation is not (yet) supported by the Platinum user-plane, so that a layered architecture with three layers had to be developed:

- *local application layer*: responsible for maintaining the local copies of the shared information objects. Each entity in this layer represents a local application which only changes its local copy if an accepted operation is received;
- *control layer*: responsible for accepting and refusing operations. It contains one or more user agent entities and a single controller entity;
- *multicast layer*: responsible for providing a reliable multicast service with order preservation. The entities in this layer can multicast and receive messages using the available user data transfer facilities (user-plane).

Figure 4 shows the shared whiteboard protocol architecture with three end-systems. One of the entities in the control layer combines the user agent and controller role; this is the case if the entity has been assigned the controller role and at the same time must support a local application. The middleware layer is not represented in Figure 4; this is the case since, from a protocol perspective, the multicast layer can directly use the user-plane (data) connections provided by the network.

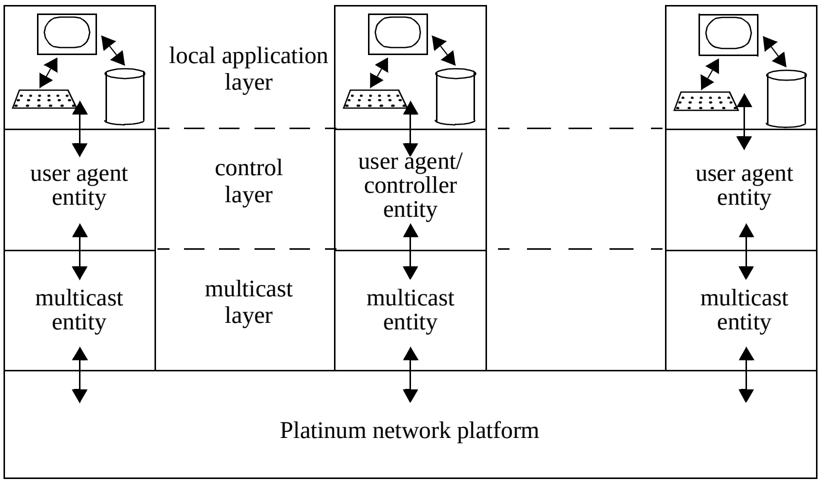


Fig. 4. Layered protocol architecture of the shared whiteboard application

3.3 Protocol behaviour

The entities in the control layer communicate through the exchange of PDUs via the multicast layer. For each interaction with a local application there is a corresponding PDU defined, except for the confirm (accepted) and indication interaction, which have the same corresponding PDU type. For example, an add-req results in the exchange of an AddReq PDU, while an add-ind or an add-cnf+ result from the exchange of an AddPCnf PDU. Figure 5 illustrates the exchange of PDUs in response to an arbitrary operation request (x-req) in case the operation is successfully performed.

A number of additional PDUs are defined to perform user agent initialization and recovery from data loss. *User agent initialization* is necessary if a shared whiteboard user is added to the current group of shared whiteboard users. A user agent is activated for the new user, and the controller entity sends a StateUpdate PDU to this user agent, conveying the current state of the shared whiteboard. This current state contains the information objects held by the local copies at that moment. The user agent returns a StateUpdateAck PDU to confirm the state transfer.

Recovery from loss is a function which seems redundant if the multicast layer provides full reliability. There are two reasons why this function is useful. First, because the multicast layer is a more generic layer than the control layer, and providing full reliability may be too ‘heavy’ and consequently too slow for many applications. A reasonable level of reliability seems to be more appropriate instead, leaving application-specific upgrading of reliability to higher level protocols. Second, the investment required for adding a recovery function in the control layer is small, since the PDUs required for user agent initialization can also be used in this case. Detection of loss is achieved by the introduction of a version number field in the PDUs that are sent by the controller in response to an accepted request

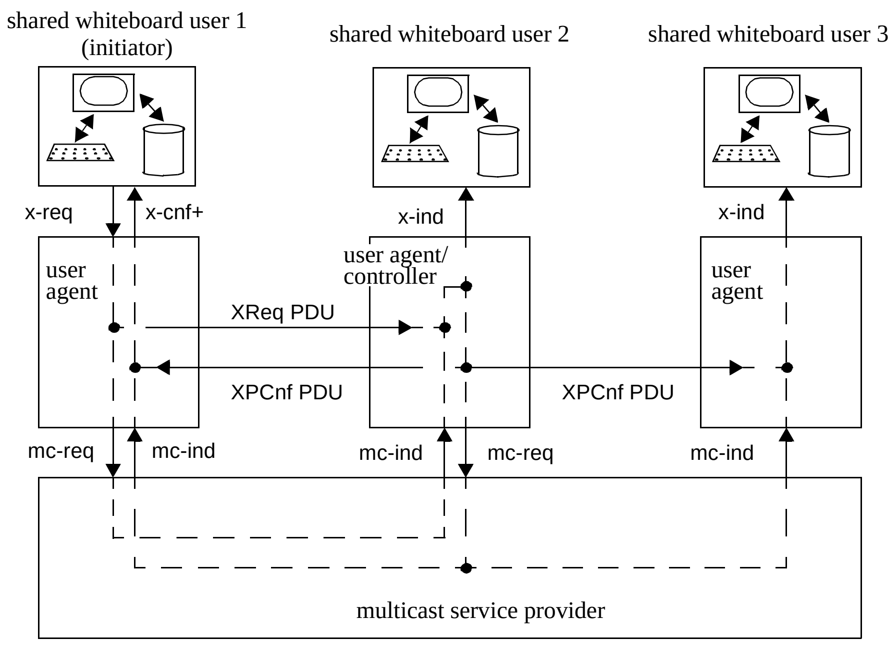


Fig. 5. Scenario for servicing an operation request

for an operation. The version number field contains a value for each information object which is referenced in the operation request; this value indicates the number of manipulations which have been accepted with respect to the associated information object. If a user agent receives a version number value which is higher than expected, this means that a previously accepted operation with respect to the associated information object has been lost, and the local copy is probably no longer consistent. The user agent then requests a state transfer, by sending a StateUpdateReq PDU to the controller. The controller responds with a StateUpdate PDU, which is subsequently confirmed with a StateUpdateAck PDU.

Depending on the PDU type or contents, the controller in the control layer either broadcasts a PDU to all user agents or it sends a PDU to a specific user agent. The entities of the multicast layer support multicast using the user-plane network connections. These connections are multi-party, which means that the end-system of the controller is in principle connected to all end-systems that host a user agent. Therefore a filtering function is necessary in the multicast layer. This filtering function is based on the exchange of a Multicast PDU with a recipient field, which specifies the actual recipients (user agents or the controller in case of the control layer) of the data conveyed in the PDU. If a Multicast PDU is received by a multicast entity and this entity does not support one of the recipients specified in the PDU, this PDU is discarded; otherwise, the data conveyed in the PDU is passed to the local recipient.

Since the connections provided by the network platform are considered sufficiently reliable, the current version of the multicast layer does not contain any reliability upgrading functions.

3.4 Software architecture

The software architecture of the shared whiteboard application is represented in Figure 6 by a class diagram. This diagram is not complete, since it only shows some essential classes of the architecture. Classes LocalApplicationEntity, UserAgentControllerEntity and MulticastEntity are specific to the application layer; the other classes are defined by the middleware software architecture.

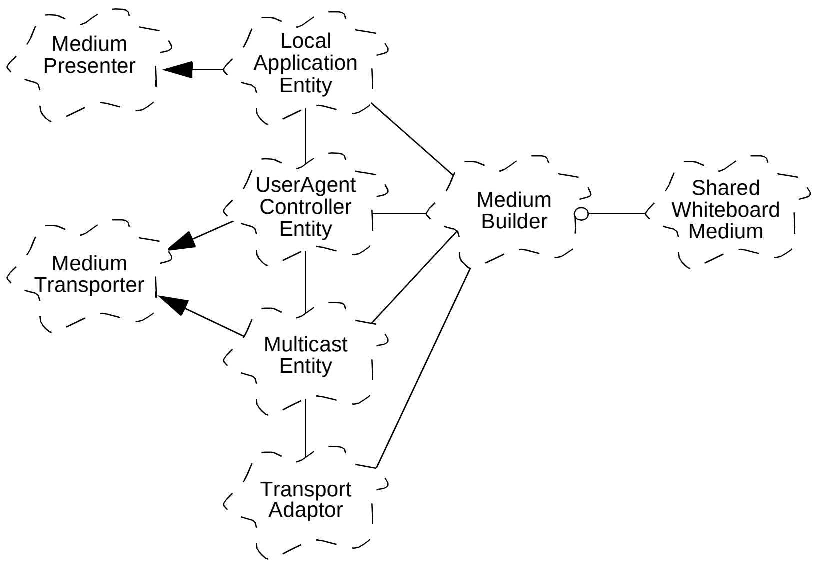


Fig. 6. Software architecture of the shared whiteboard application

LocalApplicationEntity represents a local application entity, which must control access to shared information objects and display the information objects to the user. It is a subclass of MediumPresenter, which is a general abstraction of system resources concerned with access to and management of a window. UserAgentControllerEntity represents the combined responsibility of a user agent and a controller. It is a subclass of MediumTransporter, which takes care of generation and transportation of information pertaining to some medium type. MulticastEntity represents a multicast entity and is also a subclass of MediumTransporter. MediumBuilder is responsible for building a stack of presenters and transporters, resulting in a medium of a specific type. MediumBuilder uses SharedWhiteboardMedium, which represents the shared whiteboard medium. TransportAdaptor is responsible for mapping the lowest transport interface in the stack onto the communication interface supported by the operating system.

All middleware classes mentioned here are classes which belong to the view part in the Model/View/Controller design pattern, except SharedWhiteboardMedium which belongs to the model part. The decomposition achieved in the protocol architecture is retained in the software architecture; the software architecture extends and refines this structure by also considering local aspects. For example, a more refined version of the software architecture shows a further decomposition of LocalApplicationEntity. The software architecture also shows the relationships between objects that implement protocol functions and other middleware objects.

3.5 Interactions with conference management

In order to be able to communicate with user agents, the controller must be aware of which users are participating in the shared whiteboard at each moment. Therefore the shared whiteboard application has to interact with the conference management application.

First, the addition of a shared whiteboard medium to the conference (session) causes the shared whiteboard application to be initialized and the controller to be determined. In the current design, this is not done dynamically, but rather is fixed. The initial composition of the group of shared whiteboard users will also be determined at this time, and so this information must be communicated from the conference management to the shared whiteboard application. The shared whiteboard protocol stack reside in the user-plane, whereas the conference management protocol stack works (mostly) in the control-plane. Thus, in order for the multicast layer (in the user-plane) to deliver messages to the proper destinations, it must have information about which users are attached to the medium, which is determined by conference management.

Second, the controller must be kept aware of changes in the composition of the group of shared whiteboard users. The conference management application informs the controller which users have been added to, or removed from, the shared whiteboard medium. The multicast layer group information is thus kept consistent with the state of the medium itself. The user agents do not need this information, since they communicate only with the controller and not with other user agents. The user agents learn the address of the controller when they are initialized. In this way, interaction with conference management is limited to the controller and there is no need to worry about synchronizing the information that comes from conference management among all the control layer entities.

4. Future extensions and alternatives

The current design of the control layer protocol is relatively simple. It assumes a fixed central controller that serves a group of user agents. New user agents may join this group and current user agents may leave this group, but the controller does not change during an instance of the shared whiteboard. Although this protocol has the benefit of being quite simple, there are some drawbacks. First, the fixed controller is a single point of failure. If the controller itself fails, or if the connections to the controller fail, then it simply is not possible to use the shared whiteboard further. Second, the controller cannot leave (or detach) from the medium or leave the session. This limits somewhat the flexibility of the participants and of the system as a whole. We discuss some possible enhancements below.

4.1 Dynamic assignment of controller role

A possible enhancement of this design is to allow the dynamic assignment of the controller role. The benefit of using dynamic assignment of the controller role is that the shared whiteboard application is no longer dependent of the availability of one specific end-system. With this enhancement, a controller would be allowed to leave the shared white-board, in which case its role is taken over by one of the user agents. Furthermore, if the controller fails, the shared whiteboard application would not be necessarily disrupted, since the faulty controller would be removed from the group, and the application would continue after a new controller was assigned and initialized. Finally, if performance becomes a problem, then the controllership could be transferred to an end-system that has the most processing power.

The selection of a new controller could be based on a simple algorithm, e.g., to choose the user agent with the highest or lowest network address, without involving any extra communication between end-systems. The procedures for initializing the new controller would depend on whether the controllership changes due to a ‘hard failure’ or is accomplished more gracefully. If the change of controller is due to a hard failure, then the state of the collection of information objects as seen by this new controller would become the reference state. After a more graceful change of controller, the state of the old controller could be passed explicitly to the new controller; meanwhile the old controller would refuse to accept new requests from the user agents and redirect the user agents to the new controller. Mechanisms that allow all user agents to synchronize with (the state of) the new controller would then be necessary.

4.2 Multicast reliability

Another possible extension of the current design is to add reliability functions to the multicast layer. The current multicast layer is not fully reliable and consequently the control layer contains functions to detect data loss and to repair its effects (see Section 3.3). Though this design is effective, it can be costly since it may involve the exchange of possibly large state information. For a moderately unreliable medium this design is likely to perform quite poorly.

In addition, a drawback of building reliability protocols into applications is that the cost is incurred for each application with such a requirement. This is actually the counterpart of the argument in favour of the current design presented in Section 3.3. For these reasons it makes sense to build reliability into the multicast protocol, even though this is, in itself, a formidable task. There are, however, a large number of researchers who have worked on this problem. See, for example, the references in [10].

4.3 Distribution of shared information

The use of centralised or distributed solutions to the shared information object problem presented in this paper is another important issue. The design that we have presented is clearly centralised, and has the disadvantages associated with that design: less fault-tolerance and poor scalability characteristics. However, the trade-off is between the complexity of the design and implementation effort and the requirements of the system. For the expected operating environment and for the goals of the system, such a centralised design seems to be adequate.

However, as the system moves to (much) larger scale user groups and toward high reliability requirements which may be imposed by some environments in which the conferencing application will be used, a distributed control solution will have to be considered.

Fortunately, there has been a great deal of progress in this area recently and there are a number of excellent alternate protocols available for providing distributed processes with the same view of the state of an object (see, e.g., [6, 4, 10, 5]). These protocols could be incorporated into the design shown in Figure 4 by replacing the control layer and possibly the multicast layer, without disturbing the local application layer.

5. Discussion

It is not yet possible to draw final conclusions about the flexibility of the Platinum design, since little experience has been gained with the use of the Platinum design in practice, and, consequently, the design has not been challenged by requirements for modification and extension. However, some interesting discussion points related to the Platinum design are briefly presented below.

5.1 Protocols and services

In any distributed system design, interoperability of the distributed parts of the system is a major concern. In particular when the distributed system is implemented on different hardware platforms, or is expected to evolve in order to cover additional (different) hardware platforms, a model of the distributed system which addresses interoperability aspects but abstracts form other aspects is urgently needed. Such a model is provided by a protocol architecture. A protocol architecture can be considered as an intermediate result of the domain analysis in the software development process in which interoperability aspects are explicitly addressed. The relationship between protocol design and software design is depicted in Figure 7.

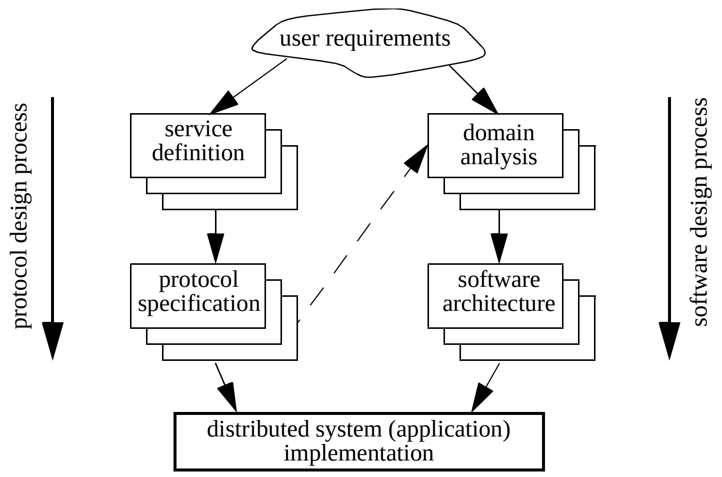


Fig. 7. Relationship between protocol design and software design

Many protocols are designed to perform multiple functions and provide multiple services. This is done for efficiency reasons but may result in complex and inflexible systems.

The Platinum design identifies micro-protocols, which are relatively simple since they perform a single function or a set of coherent functions. These protocols can be synthesized to implement multifunction protocols. By the identification of general purpose micro-protocols, a common protocol base can be established which can be used for the synthesis of several application domain-specific multifunction protocols. In addition, multifunction protocols composed from micro-protocols are easier to modify than their monolithic counterparts.

A prerequisite for the effective use of micro-protocols is a proper understanding of the interactions between these protocols. A service definition is an abstraction which facilitates a proper understanding of such interactions, even if the interacting parts are themselves composed of distributed entities. This is the case with protocols since they are composed of cooperating protocol entities. In the Platinum design, each protocol has at least two associated service definitions: the service provided to the users of the protocol, and the service assumed from a lower level protocol. Additional service (or interface) definitions may exist if ‘horizontal’ interactions are supported with other protocols. For example, the control protocol of the shared whiteboard application has horizontal interactions with the protocol of the conference management application.

5.2 Software structuring

Identification of reusable software is actively pursued in the Platinum design. By adopting object-oriented technology, reuse can be supported at different levels and in different forms. The principle of encapsulation in object-oriented software design is analogous to that of service in protocol design. It enables understanding of components without having to know their internals, and it facilitates modification of components without affecting the rest of the system. The software architecture of the middleware in particular is designed for reuse: it provides classes which can be specialized or extended in accordance to specific application domain requirements. The software architecture of the shared whiteboard application illustrates some of the possible ways in which middleware software components can be reused (see Figure 6).

Design patterns are structures and collaborations of components which have been proven successful in some application domain (see [8]). Adopting design patterns during the design process can be seen as an effective high level reuse of software solutions (in this case, software architectures). The Platinum middleware design is based on the Model/View/Controller design pattern. This design pattern has been originally identified in the design of graphical user interfaces, but also proved to be effective in the middleware software design process.

5.3 Middleware platforms

The Platinum middleware is not a true middleware according to the criteria mentioned in, for example, [1]. The current middleware implementation, which has been coined the *MediaBuilder*, only supports one operating system type (Windows NT) and one network type (accessed via the TS-API, see Section 2). Future research, however, may address other operating systems and networks. Hence, one of the first challenges which the Platinum design has to face may well originate from the designers themselves, who want to extend the support provided by the middleware.

The Platinum design distinguishes itself from other developments in two respects. First, it builds on a multiparty network based on ATM technology. As far as we know there are currently no commercial platforms for multimedia application development with these characteristics. Second, the Platinum design is not limited to the client-server paradigm. Therefore it also differs from developments such as, e.g., CORBA. The integration of the Platinum design with frameworks such as OMG CORBA and ITU-T T.120 is in principle possible and will be investigated in future.

References

[1] P. A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*, 39(2):86–98, Feb. 1996.

[2] G. Booch. *Object-oriented analysis and design*. The Benjamin-Cunnings Publishing Company, Inc, California, USA, second edition, 1994.

[3] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, and R. Tomlinson. MMConf: an infrastructure for building shared multimedia applications. In *CSCW 90 Proceedings*, pages 329–341, Oct. 1990.

[4] D. Dolev and D. Malki. The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70, Apr. 1996.

[5] A. Fekete, M. F. Kaashoek, and N. Lynch. Implementing sequentially consistent shared objects using broadcast and point-to-point communication. Technical Report MIT/LCS/TM-518, Laboratory of Computer Science, Massachusetts Institute of Technology, Cambridge, USA, June 1995.

[6] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: a fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, Apr. 1996.

[7] J. Palme and T. Tholerus. SuperKOM - design considerations for a distributed, highly structured computer conferencing system. *Computer Communications*, 15(8):509–516, Oct. 1992.

[8] D. C. Schmidt. Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM*, 38(10):65–74, Oct. 1995.

[9] H. J. Stuttgen. Network evolution and multimedia communication. *IEEE Multimedia*, 2(3):42–59, Fall 1995.

[10] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: a flexible group communication system. *Communications of the ACM*, 39(4):76–83, Apr. 1996.

[11] K. Watabe, S. Sakata, K. Maeno, H. Fukuoka, and T. Ohmori. Distributed multiparty desktop conferencing system: MERMAID. In *CSCW 90 Proceedings*, pages 27–37, Oct. 1990.