ABSTRACT

Message routing and state management are the key functions that must be supported by every collaborative system. The message routing mechanisms enable distribution of information in the system. The state management is responsible for maintaining the data essential for the system operation and control of their modification. The purpose of this paper is to investigate the generic needs of the collaborative systems as far as the message routing and the state management are concerned. It discusses various models and architectures of the collaborative environments in this context. Also, the paper describes in detail how the message routing and the state management issues are addressed in the design of the Tango2 system currently being developed at Northeast Parallel Architectures Center at Syracuse University. The analysis is focused on the Tango2 server, one of the main components of the system.

MESSAGE ROUTING AND STATE MANAGEMENT IN COLLABORATIVE ENVIRONMENTS

BY
LUKASZ MICHAL BECA

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate School of Syracuse University

December 1997

Approved:_____

Marek Podgorny

Date:_____

Copyright 1997 Lukasz Michal Beca

All rights reserved

COMMITTEE APPROVAL PAGE

ACKNOWLEDGEMENTS

Many thanks to all members of the Tango team:

Gang Cheng

Tomasz Jurga

Konrad Olszewski

Piotr Sokolowski

Krzysztof Walczak

Also, I would like to thank my NPAC colleagues:

Grzegorz Lewandowski

Pawel Roman

for many discussions that resulted in new ideas.

Funding for Tango project was provided in part by the Rome Laboratories and National Science Foundation.

Special thanks to my advisor Dr. Marek Podgorny.

TABLE OF CONTENTS

1 Introduction 1

2 Collaboration Models 2

2.1 Asynchronous Collaboration 3
2.1.1 State Management 5
2.1.2 Message Routing 8
2.1.3 Example: HyperNews 9

2.2 Synchronous Collaboration 10
2.2.1 State Management 12
2.2.2 Message Routing 15
2.2.3 Example: Corona 17

2.3 Other Collaboration Models 18

3 Collaboration Architectures 19

3.1 Centralized Systems 20
3.1.1 State Management 20
3.1.2 Message Routing 22
3.1.3 Example: Habanero 22

3.2 Server Networks 23
3.2.1 State Management 25
3.2.2 Message Routing 26
3.2.3 Example: Internet Relay Chat Protocol 27

3.3 Fully Distributed Systems 29
3.3.1 State Management 30
3.3.2 Message Routing 31
3.3.3 Example: Open DVE 32

3.4 Other Architectures 34

4 Tango2 System 35

4.1 Overview 36
4.1.1 Purpose 36
4.1.2 Functionality 36
4.1.3 Architecture 43

4.2 Basic Concepts 45

4.3 State Management 49
4.3.1 Components of the System State 49
4.3.2 State Modification 59

4.4 Message Distribution 67

4.4.1 Types of Messages 67
4.4.2 Channels of Communication 68

5 Tango2 Server 70

5.1 Tango2 Server Architecture 70
5.1.1 General Structure 70
5.1.2 Room Structure 72
5.1.3 Flow of Messages 76

5.2 Tango2 Server – Analysis and Design 80
5.2.1 Object Model 81
5.2.2 Dynamic Model 89

6 Conclusions 90

7 Bibliography 94

8 Vita 97

LIST OF ILLUSTRATIVE MATERIALS

Figure 1: Collaboration in the asynchronous systems 4

Figure 2: Collaboration in the synchronous system 11

Figure 3: Centralized system architecture 20

Figure 4: Server network architecture 24

Figure 5: Distributed architecture 29

Figure 6: Architecture of the Tango2 system 44

Figure 7: Allowed transitions of the lock mode 56

Figure 8: Tango2 server architecture 70

Figure 9: Architecture of a Room module 72

Figure 10: Routing of incoming messages 77

Figure 11: Room broadcast 78

Figure 12: Session broadcast 79

Figure 13: Unicast 80

Figure 14:Part of the object model of the Room Control module 83

Figure 15: Part of the object model of the Room Control module 85

Figure 16: Object model of the Message Distributor module 87

Figure 17:Dynamic model of the session join operation 90

1 INTRODUCTION

In recent days, we can observe rapid development of the Internet. Infrastructure of this global network grows and every year new services are available. This unprecedented growth was caused by the introduction of the WWW technologies that are still driving forces of the Internet expansion. The popularity and the ubiquity of the Internet make it ideal basis for construction of the collaborative environments. In fact, the creation of a collaboration environment for the geographically dispersed users was one of the reasons for creation of this network of networks. Collaborative tools such as the electronic mail have been present in this environment for a long time. However, recently many groups of the Internet users declared need for more sophisticated collaborative tools. Large number of new ideas such as Computer Supported Cooperative Work emerged. Such circumstances caused an explosion of various systems that support scientific collaboration, distant learning, videoconferencing and many other forms of collaboration. Also, the huge popularity and increased reliability of computer networks drew interest of the commercial companies that now support on-line customer service as well as use the Intranets as a medium for the distribution of the information and a tool for the group work.

This paper discusses two functions that are crucial for the construction and the operation of the collaborative environment: state management and message routing. State management concentrates on the maintaining and modification of the data that are essential for the proper functioning of the system. Message routing focuses on the definition of the channels of communication in the system and types of data transferred using those channels.

The purpose of this paper is to analyze the state management and message routing issues that arise in various models and architectures of the collaboration environments. It also describes how those issues are addressed in the design of Tango2 - the collaborative system currently developed at Northeast Parallel Architectures Center at Syracuse University.

The document starts with the discussion of various collaboration models – Section 2– and architectures – Section 3. This part of the paper tries to identify typical problems that emerge in those models and architectures. It also tries to identify the solutions to those problems and show how they are employed in the existing environments.

Section 4 describes the Tango2 collaborative system. It specifies the functionality offered by the system and its impact on the state management and the message routing. This section describes in detail the components of the system state and actions that result in the state change. It also enumerates the channels of communications provided by the system and types of data transferred through those channels.

The last part of this paper describes design of the Tango2 server. Section 5 shows how the issues described in the preceding sections influence design of the main elements of the collaborative system.

2 COLLABORATION MODELS

This section describes how the state management and the message routing differ in various collaboration models. As a division criterion the synchronicity of the collaboration is taken. This factor greatly affects the properties of the system. Another important factor – architecture of the system – is discussed in Section 3. The choice between synchronous and

asynchronous model of collaboration determines what state information the system should store and what are the means of modifying this information. Also, the synchronous and asynchronous systems impose different constraints on the message distribution in the system. Those two types of environments have different needs that should be taken into account when designing collaborative system. For each class we will try to distinguish generic requirements for state management and message routing so that they apply to every environment that belongs to this class. Each case is illustrated by an example of the real system.

2.1 ASYNCHRONOUS COLLABORATION

Asynchronous collaboration occurs when users can exchange information without being simultaneously in contact with the system. Figure 1 presents an example of the asynchronous collaboration. User1 terminates connection with the system before user2 connects to the system but the provided functionality enables exchange of the information between both users. The time boundaries of collaboration session are not defined. The collaborative session lasts as long as the system is running and there is no strict timing requirement for the data delivery. The reliability is more important than the speed of information transfer. The users do not expect quick responses from the other participants but they want to be sure that the message will finally reach the destination.

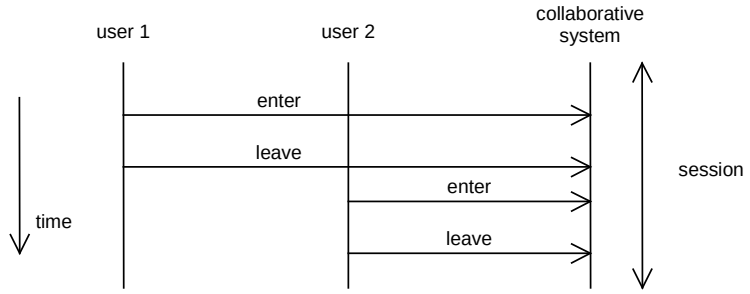

Figure 1: Collaboration in the asynchronous systems

Asynchronous collaboration may be set up in one-to-many and one-to-one pattern. One-to-many model is usually constructed around shared content. The shared content is a set of data that is accessible to the users of the system. Examples of such content are the document annotations in case of HyperNews system [1] or the articles in case of the Network News Transfer Protocol [2]. The users connect to the system, modify shared data e.g. add new article or new annotation to the document and leave the system. When the other users connect to the system they can observe that the data were modified. After reading the new data, they may introduce new modifications. Such approach requires some kind of shared content management. Limits on size as well as lifetime of the data should be defined especially if the service is accessible to large number of users. On the other hand, shared content should be preserved till every user gets update of the changed state. Moreover, the various restrictions of the access to the shared data may be introduced. Using authentication mechanisms, the system may distinguish the users that may read, add or modify shared data.

One-to-one model does not rely on the shared content but rather on moving data from one user to another. Such model is employed in the electronic mail system specified in the [3].

2.1.1 STATE MANAGEMENT

In order to work properly, the system must maintain the data that describe its state. The state of the asynchronous collaborative system may be divided into two categories: the collaboration framework state and the shared content state. The collaboration framework state describes the condition of the infrastructure that enables information exchange. The shared content state describes condition of the exchanged data.

The framework state of the asynchronous system may be composed of several types of information. Usually following data is maintained:

- user list
- access log
- group member list

The user list is the list of people that use the system. It contains the identification data sometimes complemented by the password necessary for the authentication process. Some systems store the access log. The access log contains information about parts of the shared content accessed by the user. Such information may be helpful in determining what data is new for the users. The systems that have large number of users may support user groups. The members of one group usually have common interests and access the same data. Support for the user groups requires storage of member lists for each group.

The framework state of the asynchronous system should be persistent because collaborative sessions last for unlimited time. Usually it is stored on the disk. Such solution provides sufficient protection against various types of failures.

The collaborative framework state may be changed by actions of the users. Those operations are:

- register and unregister user
- subscribe and unsubscribe from the group
- access document

The register operation tells the system that the user will use the system in the future. As a result, user name and optionally other information such as password and address is added to the user list. From now on the user can collaborate with other system users. Unregister operation has the opposite effect. The system removes the information about the user from its user list. The subscribe operation is used if the user wants to have access to the specific group of interest. Such operation results in adding the user to the specific group member list. The unsubscribe operation removes the user from the group member list. The framework state may be also modified by accessing the shared data. The systems that maintain the access log will mark accessed data as read by the user. If the system provides different access rights for different users some of the enumerated operations may be available only to the limited set of users.

The system usually stores following information as a shared content state:

- shared documents e.g. articles, annotations, electronic mail
- information describing the shared content such as username of the author of the document or timestamps

The shared documents are the shared content itself. They represent the data that the users create and work on together. Interesting modification is the storage of the pointers to the actual data as in the case of the HyperNews system that may store URLs of the resource instead of the resource itself. The information about author of the shared document or of the shared document modification is very important. It says who is responsible for the specific piece of the shared data and to whom all the questions and remarks may be directed. Timestamps are useful for the partial retrieval of the information, for example the user may request the documents older than the specified date. They are also necessary for the storage management – enable removal of older documents from the system database.

The shared content may be subject of adding new data or changing of the data. Asynchronous systems usually prefer adding new data instead of changing. The former approach is much safer and guarantees that interested users will have access to all data that was created.

Another important issue of the shared data management is the way the users find out about data changes. Two solutions of this problem exist [4] :

- notification
- polling

The systems that implement the notification, send a message to the user whenever the state of the shared data change. For this purpose access log may be maintained for each item of data, recording all accesses to this resource. Notification is sent whenever new information is added or shared data is modified. Notification may be sent to all system users or only to the group interested in particular data.

The polling puts the burden of learning about data modification on the user side. The software run by the user, periodically asks the system whether something changed in the state of the shared data. Again, access log may be useful, because it allows the system to determine if the user accessed particular data after or before latest data modification. The same functionality may be provided by the timestamps added to the stored information. In this case, the user that wants to retrieve the unread data should specify date of the last connection in the request. As a result, the information with timestamp grater than specified date will be transferred as not accessed.

2.1.2 MESSAGE ROUTING

Asynchronous collaboration may be implemented using different routing models. Most often TCP/IP protocol is used because of its reliability. The reliability is the most important factor in the asynchronous collaboration systems. Time is not so critical and messages are allowed to be delayed but finally they must arrive. Communication between user and the system is maintained by exchanging following information:

- the user sends to the system
  - added information
  - subscribe/unsubscribe notification
  - polling message
- the system sends to the user
  - retrieved data
  - new message notification

2.1.3 EXAMPLE: HYPERNEWS

As an example of asynchronous collaborative environment we will investigate HyperNews system. HyperNews [1] system developed at NCSA, supports discussion on the Word-Wide Web. It allows users to add comments and responses to the existing WWW pages. The discussion is formed around a 'base article'. To this article the comments and the responses are added. The comments to already existing responses are possible so the structure of the discussion resembles a tree. Several forms of comments are recognized, including HTML and URL that points to the place where the response is stored. As the discussion develops, sometimes it is desirable to reorganize it so that structure is clearer. For this purpose, creation of the new base article is possible or response subtree can be moved to the other location. Such operations are permitted only for administrators (owner of the base article becomes an administrator for the article and responses). The system has also support for the hierarchical groups of users with different rights.

Users may perform following operations:

- becoming a member
- reading articles and responses
- creating base articles
- adding responses
- deleting/moving responses
- subscribing

All those operations may be restricted to the different degree. Several types of constraints are supported:

- no restriction applies to the operation
- all members of the group may execute the operation
- only administrator may execute the operation

HyperNews supports notification. Whenever new response is added to the discussion tree, e-mail is sent to all authors of ancestors of the new response. Also, each article and response has its own list of subscribers, which is used to send notification.

The system is implemented as a set of CGI scripts, that return standard HTML documents. It means that any Web client without modification may use the system.

2.2 SYNCHRONOUS COLLABORATION

Synchronous collaboration provides tools for cooperation of users that are present simultaneously in the system. Simultaneous presence is indispensable condition for collaboration. Such approach raises several issues. Figure 2 presents an example of the synchronous collaboration. The users connected to the system may send the events one to each other. In this way the actions of one user are immediately visible to the other user. The collaborative session has strictly defined time boundaries and exists as long as the users can exchange the data using the system.

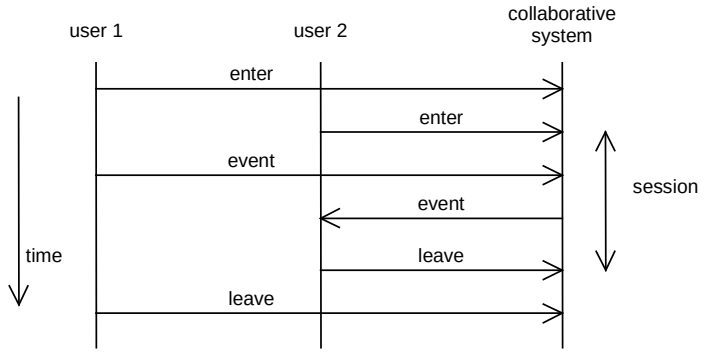

Figure 2: Collaboration in the synchronous system

The user working in the synchronous collaborative environment interacts with other users. Usually those activities are performed in groups. Participant must know with whom he or she works and what kind of access he or she has to the system tools and resources. The system should provide this information. Usually, user will see list of the participants as in the Tango system [5] or graphical objects that represent other participants as in the CSpray system [6]. Such mechanisms provide collaborative awareness, that helps users to define their place and role in the collaborative environment. Different models of collaboration require different kinds of awareness. The scientists that work together on the same research project in shared environment will have different needs than a student that participates in distant learning session. Those mechanisms may be extended to enable users to express their focus on the particular item of the shared data e.g. point on the fragment of the document or highlight the part of the drawing. This may be useful when the group works on the shared document or in distance learning applications.

The participants should get the updates when the state of the system changes e.g. user leaves or joins a group. Also the new user, when joining collaborative group should have the possibility to receive the information about the state the system. Action of one user should

be visible to the other users as soon as possible. Such requirements introduce important constraints on message distribution mechanisms. The synchronous collaboration systems try to keep all the clients in consistent state and provide various means of synchronization for this purpose.

Various collaborative tools have different requirements as far as quality of the communication channels are concerned. Applications that handle multimedia streams e.g. video-conferencing systems are very delay-sensitive but do not require perfect reliability. On the other hand, applications like whiteboard or chat need very reliable communication and not necessarily very quick. The obvious conclusion is that those two groups of applications should be handled in different way.

2.2.1 STATE MANAGEMENT

State management in synchronous collaboration systems is much more complex than in the asynchronous collaboration. Synchronous environment must enable direct communication between users and amount of information needed for that purpose is larger. We will divide the system state into two components: the collaboration framework state and the collaborative applications state.

As in the asynchronous collaboration, the collaborative framework is defined as the infrastructure that enables communication between the system users. The collaboration framework state contains some or all of the following elements:

- authentication information
- current sessions

- current users
- current applications

Authentication information is used to control the access to the system. Some systems provide services only to the selected group of users. They employ the user lists with the authentication information such as username and password for distinguishing valid users. Other systems introduce limitations in access to various resources by assignment of different rights to different users. Such assignment may be also controlled using list of users with the entries with specified access rights.

Another component of the collaboration framework state is the information about users and applications: how they interact with the system and how they interact with each other. These data define dynamic structure of the collaboration. The information about sessions tells the system how the users are grouped and what channels of communication should be established so that the participants of the session may exchange data. Using this information, system may also send updates to all participants of the session about events that affect state of the session. The access to the session may be limited, for example Internet Relay Chat protocol defines three types of sessions: private, public and invite only. The participants of the session may have different status such as observer or administrator. The users with different statuses have access to different set of actions. The state of the collaboration framework may be modified by the following operations:

- user actions
- administrator actions
- system failures

User actions usually change dynamic state of the system. Participants may enter and leave the system, run and stop the applications, join and leave the sessions. All those operations must be strictly controlled by the system in order to keep the framework state in a consistent form. Information about actions of the user should be distributed to other users to ensure the awareness. Some operations such as modification of accepted user list should be restricted to the system administrators. Finally, various types of failures may affect the operation of the system. Parameters of the Internet connections vary and communication links are not reliable. Also the applications that communication with the system may start to behave unpredictably. The collaborative framework should maintain the consistency of the state even in such situations. One of the possible solutions to this problem is constant polling of the system elements. The element that does not respond in the specified amount of time is removed from the system and the state is updated accordingly.

Another component of the synchronous collaboration system is the state of the collaborative applications. We are particularly interested in the shared state of these applications because it enables communication between the application users. The shared state of the collaborative applications may be implemented in two ways:

- shared state stored by the system
- state stored by applications

In the first case, the system maintains the shared data and the applications must ask the system for access to these data i.e. shared state of the applications is under control of the system. In such situation various access policies may be introduced. e.g. it may be necessary to lock the data item in order to modify its value.. In this way, floor control may be easily

provided. Also, system control over shared data guarantees consistent state of all applications.

In the second approach, applications store their state by themselves and send notifications to other application whenever the local state changes. This approach is used by the shared display systems as HP SharedX [7]. It is much more difficult to provide state consistency in all the applications, but this solution is much easier to implement. The system only passes event messages and task of the share state management is imposed on the applications.

Some systems have capability to save the state of the collaborative applications so that it may be retrieved later e.g. Futplex system [8] provides tools for work with the shared documents that may be saved and restored for further work. Such functionality puts those systems on the border between synchronous and asynchronous collaborative systems.

2.2.2 MESSAGE ROUTING

Synchronous systems use whole array of collaborative tools. Some of them have different needs as far as properties of used data streams are concerned. Chat applications that use text stream do not need very quick response time, but reliability must be guaranteed. On the other hand, multimedia streams e.g. audio and video are delay-sensitive and in this case reliability may be sacrificed. Such customization of service was implemented in several systems. One of them is InVerse [9] system that distinguishes several kinds of data streams and enables customization of the data stream handlers for the specific needs. Following data stream types are supported: object updates, application data, bytes, audio and text. Incoming

data packets are prioritized based on their real-time requirements and receive different kind of service.

Apart from different characteristics of the data streams, different models of collaboration need different models of communication. Several basic communication channels may be distinguished:

- one to one
- one to many
- many to many

One to one channel is equivalent to the telephone call. It provides communication between two applications used by different users. This channel is used when the transferred information should be accessed only by the particular user in the session. This channel may be employed by the professor in the distant learning session when evaluating the student. Information about grade of the student should not be accessible to the other student participating in the same session. This channel of communication may be also used by the control messages e.g. during authentication procedure.

One to many channel may be used when users in the collaborative group have different status e.g. professor and students, shared scientific instrument and readout data receivers. The source sends data to the receivers or a user changes shared data and other users observe this activity. Such channel of communication is provided by the Corona system [10]: the scientific instrument connected to the system sends the readouts to all interested users. Many to many model is employed, when close collaboration is necessary. Each member of the

collaboration group has the same status and may perform the same actions. Each participant may be the source of data or may modify shared data.

2.2.3 EXAMPLE: CORONA

Corona is a communication service developed as a part of the Upper Atmospheric Research Collaboratory (UARC) project. Corona supports group collaboration. It's main purposes are to provide space scientists with tools to effectively view and analyze data gathered by various remote instruments installed in Greenland and to create environment for collaboration between distributed communities of scientists.

Corona provides services based on two models:

- publish/subscribe
- peer group

Publish/subscribe service enables entity called publisher to send data to large number of recipients (subscribers). This is basically anonymous form of communication. The publishers know about all the subscribers but subscribers do not know about each other. Such model is preferred for distribution of the instrument readouts. In peer-group model all group members are aware of each other. They may exchange data with each other with some limitations. Limitations are imposed in form of different roles: observers and principals. The observers may only receive data while principals may send and receive data. This model is appropriate for the collaborative work of small groups of scientists. The system provides failure detection. If the confirmation does not come, messages are retransmitted and if it does not have desired effect crashed participant is removed.

2.3 OTHER COLLABORATION MODELS

Significant number of systems cannot be classified as a synchronous or asynchronous. Such systems usually merge those two models of collaboration. They provide the possibility to record the status of the shared work so that participants can continue it over longer period of time or they have capability of recording the sessions so that they may be re-played.

First approach is used in Collaboratory Builder Environment (CBE) developed at the University of Michigan [11]. CBE uses rooms as the high-level grouping mechanism for users and applications. Rooms may contain applications, users and data objects.

The system provides persistence of data in the rooms, so that state of the sessions may be stored. Rooms maintain the data even when no user is present. Using this mechanism, the applications may restore their state from the previous sessions.

A little different approach is applied in the Multi-User Dungeon (MUD) [12] systems. Those distributed environments for role-playing enable players to interact in the synchronous manner while exploring the virtual worlds. They also allow the user to save his or her status in the game and to restore it whenever the user logs in again. Players may also send notes one to each other.

Another solution was implemented in the TechTALK system[13], which incorporates mathematical tools like Maple or Matlab in the collaborative environment. Using tools supplied by the system the users may share Maple and Matlab sessions. Each user may send command to the mathematical engines and all participants may observe results of the

computations (including pictures). A logging facility permits to record a session and replay it as a class lesson in the electronic tutorial.

Finally, already mentioned Futplex [8] system enables users to work together on the shared HTML documents and save the results of their work.

3 COLLABORATION ARCHITECTURES

The users of the collaborative systems may be distributed over vast areas. Therefore, the systems must provide the means of communication for the users that work on remote sites. In this task they rely on the services provided by the Internet. Various architectures are employed in order to allow for various needs of the users of those systems. Generally they may be divided into three groups:

- client-server - centralized
- server networks - partially distributed
- peer-to-peer - fully distributed

Many clients connected to one server depict client-server architecture. Server networks operate as a set of interconnected networks that handle requests of many clients. Finally in peer-to-peer solution, servers are eliminated and all data is send directly to the clients. Chosen architecture greatly affects how the state of the system is stored, updated and synchronized. It also has crucial impact on message distribution mechanisms. Following sections address various issues concerning the state management and the message distribution that occur in those architectures.

3.1 CENTRALIZED SYSTEMS

System is centralized when one entity - central server - provides services to all clients. This architecture is illustrated in the Figure 3. The user must know the location of the central server in order to establish connection and start collaboration.

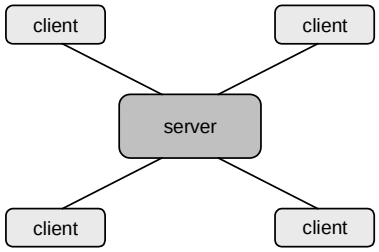

Figure 3: Centralized system architecture

All the messages are send to one point and they are stored there or redistributed to other clients. The central server must be very reliable because any failure of this application brings the entire system down. On the other hand the server may be run in the controlled environment with reserved access to all necessary resources. The requirement of passing all the messages through one point makes the state management easier but poses serious scalability problems. When the number of the users becomes large, the system gets overloaded and quality of service decreases. This architecture is unacceptable for applications that transfer large amounts of data that are delay sensitive.

3.1.1 STATE MANAGEMENT

State of the centralized system is kept in one place and therefore its consistency is easily maintained. Also, various techniques of the synchronization, like floor control, may be

implemented because one application – central server - stores and has access to all collaboration framework and application shared state information.

Change of system state usually follows one pattern. A user that wants to perform any action must send request to the central server. If the request is valid, the state is changed on the central server. Depending on the implementation, the information about change is distributed among all the clients or only the user that started the action is notified about action final status - successful or failed.

In centralized architecture it is possible to implements shared objects, that are accessed by the collaborative tools. Such shared object may store the state of the application shared with the other applications. The server may control access to the shared objects by means of object locks. The application that wants to change the value of the shared object must first lock the object exclusively, perform the modification and release the lock. Such solution guarantees that all instances of the application used by different users will be in the same condition during the collaboration session. The idea of shared object solves also problem of the latecomers. The users that joined collaborative session after it was active for long time want to know what happened while they were absent. The central server may transfer the content of the share object to the new application so that it will be synchronized with the others participants of the session. Shared object concept is very useful in building synchronous environments that provide shared workspace where strict synchronization of the collaboration tools in necessary.

Also, shared content management is the easily implemented using centralized approach. Such approach is used by the Futplex system in the management of the access to the shared

HTML documents. Storing shared documents in one place guarantees their consistency. Similar locking techniques as for the shared objects may be applied for access control.

3.1.2 MESSAGE ROUTING

Clients of the centralized system pass all the messages to the central server. The server, using stored information about state of the system, may send events to the selected client, specific group of clients working in the same session or may broadcast events to all users connected to the system. The server is directly connected to all the clients what makes message routing easier. However, when the number of the users is large the server may have difficulties in handling all the requests. As a result, the data transfer through the server is delayed and throughput of the system decreased. Another disadvantage of the centralized approach is the necessity to send the data over long routes even when the users are very close. In such case direct communication would be much more effective than accessing the server running on the remote site.

3.1.3 EXAMPLE: HABANERO

NCSA Habanero [14] is an environment that enables creation of the collaborative environments. System provides session management tools, so that the users may start and joins sessions as well as retrieve information about other participants. Habanero also supports shared state and shared actions that are distributed to all the clients or to the selected group of clients.

Habanero is implemented using centralized—client-server architecture. The server routes all the messages between applications, transfers shared state and actions, provides

arbitration and synchronization mechanisms. The state of the tool is sent by *marshalling* and *unmarshaling*. Those operations are responsible for serialization i.e. encoding and decoding of the shared objects. Encoded objects are sent using the channels of communication provided by the server and decoded by the receivers. Using entity called arbitrator, which resides on the server, some limits on the user actions may be imposed. Several types of the arbitrators are supported. The arbitrators manage locks that define the types of actions permitted to execute by the users.

3.2 SERVER NETWORKS

The users of the collaborative systems may be represented as a set of communities. The members of one community work at the same site and may even share the same network infrastructure. Different communities are remote one to each other, placed in different parts of the country or of the world. Server network approach provides each community with its collaborative server that satisfies the needs of local users as well as allows shared work with members of the remote groups. The Figure 4 presents example of the system that uses server network architecture. The groups of clients are connected to the servers. The servers are interconnected so that the communication between the users on the remote sites is possible. Such approach provides better scalability than the centralized system.

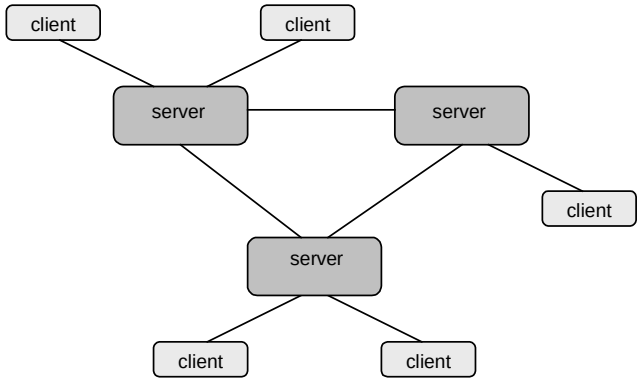

Figure 4: Server network architecture

The server network is an interconnected group of servers. The servers may be connected permanently or periodically – when the need for the data exchange occurs. The former solution is applied in the Internet Relay Chat [15], which offers synchronous collaboration. The latter is employed in the Network News Transfer Protocol, which supports asynchronous collaboration. The users, in order to enter collaborative environment, must connect to one of the servers. The nodes of the server network exchange data so that the collaboration between the users connected to the different servers is possible.

The server network has better scalability than the centralized systems but introduces additional complexity to the state management. In order to work properly, the state of the system must be consistent, so whenever user changes the state of the local node of the server network, e.g. posts news article or enters IRC channel, information about this change must be distributed to other servers.

Also, the message distribution is more complicated. The message routes must be established for data exchange between users connected to different nodes of the system.

3.2.1 STATE MANAGEMENT

One of the most important tasks in server network architecture is the system state distribution. The changes of the state that take place in one node should be distributed to others, so that the participants connected at any node can access the same information. This problem is solved by establishing the protocol of communication between the servers. Various mechanisms were developed to assure consistency and proper state synchronization.

For example, backbone of Internet Relay Chat Protocol is formed by the set of the servers. Servers may be connected only in the configuration of spanning tree. If the cycle in the network is detected, the connection that formed the cycle is terminated. Each server acts as a central node for the whole network of the servers. The spanning tree configuration ensures that each server will be updated only once whenever change of the state occurs e.g. new user enters a system.

Protocol that is a base for Internet news system - Network News Transfer Protocol deals with the state update problem in different way. The servers that exchange news articles may choose which articles they want to obtain one from each other. Following procedure is executed. Whenever the server wants to get new news or to send news articles it must contact other server using NNTP. After the connection is established, the query about new article groups is generated and if there are any, those groups are created on the client i.e. the server that initiated the connection. In the next stage, the client requests list of the articles that were send to the newsgroups on the server. The client may request lists from all newsgroups supported by the server or only from the chosen groups. Those articles that are not present on the client are transferred and added to the client newsgroups. In the last

stage, the client offers its recently received articles, which then may be transferred to the server. Such procedure ensures that only messages needed by both sides of the connection are transferred.

Another way to distribute the state is called flooding. The server that supplies the articles to the other server sends all its news articles. It is the task of the receiver to discard the duplicates and unwanted messages. This method is much simpler but also inefficient especially when updates are performed frequently.

3.2.2 MESSAGE ROUTING

Message routing in the server network architectures is more complex than in centralized systems. Following patterns of communications may be distinguished:

- client - server - client
- client - net of servers - client

First scenario occurs, when the collaborative server handles requests of the users from local community. All users from the same community are connected to the same node of the server network, so the collaborative work may be handled by only one node of the system.

Second scenario occurs when participants of the collaborative sessions are members of remote communities and are connected to different nodes of the server network. This issue may be addressed in different ways. The server that receives message for the user that is not a member of local community uses collaborative framework state data to determine how to reach the destination user. For this purpose, some kind of routing algorithm is necessary that determines the path from one user to another through server network. For example, Internet

Relay Chat protocol has a backbone structured as a spanning tree so there is always one path connecting different users. Another approach is taken in the Simple Mail Transfer Protocol. The servers that support this protocol try to connect directly to the SMTP server, that maintains mail service for the destination user. Yet another solution is presented in NNTP specification, where the state of the shared content is distributed to all nodes in the server network until finally it reaches the server that handles connections with the destination user.

3.2.3 EXAMPLE: INTERNET RELAY CHAT PROTOCOL

Internet Relay Chat protocol is a the text-based conferencing environment. It provides users with tool for synchronous communication. IRC provides mechanisms for message distribution and session management.

The infrastructure of the IRC is formed by the network of servers. The server is a central point to which the clients and other servers may connect. The clients connect to the IRC server to join chat session. The servers connect to the other IRC servers to create a network. The only permitted configuration is one of the spanning trees.

All servers must know following information about client:

- nickname of the client
- real name of the host where the client is executed
- username of the client on that host
- the server to which the client is connected

This information is necessary for efficient message distribution and session state management. Special users – operators - are allowed to perform administration and maintenance tasks. They may execute commands that are not accessible by the other users.

In IRC system, users are grouped in channels. Each channel has its own name, by which it may be referred in the commands e.g. joining channel. Messages emitted by the client in the channel are distributed to all clients in the channel. Channel is created when the first user enters and it is destroyed when the last user leaves. The system supports two types of channels: distributed channels and local channels. The former are known to all servers in the network and may be joined by any client in the system. The latter are assigned to the IRC server and may be joined only by users connected to this server.

The IRC system supports various communication modes:

- one to one
- many to many
- one to all

One to one communication is usually used by the clients. The IRC routes the messages through the shortest path between two nodes in the spanning tree.

Many to many mode is used for conferencing. The user may send message to the list of users, to a group or to users on specific host or server. The most common way is sending messages to a group called channel, which distributes messages to all clients in the group. All messages are routed through the shortest path in the spanning tree. If several recipients are

connected to the same node only one message is sent to this node. Then, the server distributes the message to all the recipients.

One-to-all mode enables client or server to send messages to all client, servers or both. Such mode of communication is necessary for the system state distribution and maintenance. Usually it is used when the state information changes e.g. new client joins the channel or the channel mode changes.

3.3 FULLY DISTRIBUTED SYSTEMS

Fully distributed systems provide means of direct communication between clients without any intermediary entities. This approach ensures the best scalability especially when message sending is implemented using IP multicast services. Figure 5 presents illustration of the distributed architecture. The whole system is composed only of clients.

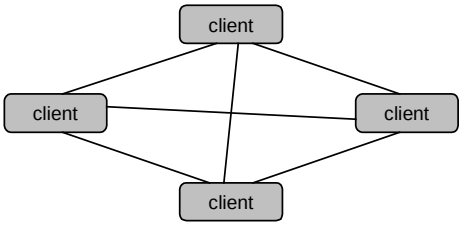

Figure 5: Distributed architecture

The state management in the peer to peer architecture is distributed over all clients of the system. Such approach poses many synchronization and consistency problems. Because of that, the fully distributed systems have limited functionality as far as the session management is concerned. It is very difficult to implement complex models of collaboration using this approach. Peer-to-peer architecture is preferred by environments that provide

multimedia collaboration e.g. video conferencing because it assures the smallest delay in the data distribution process.

3.3.1 STATE MANAGEMENT

There are two important problems that effect state management in peer-to-peer architecture:

- state synchronization
- session establishment

First problem occurs when the user changes the state of the system locally e.g. starts new collaborative tool or leaves the session. All users collaborating with this user should be notified about changes in the system in order to update their copies of the state information. This task is difficult to achieve using connectionless IP multicast or UDP communication, because they are unreliable [17]. One of the solutions of this problem is to maintain only approximate state of the system and accept the delays in the state distribution. Another solution is to keep reliable TCP/IP connections between all users as in the CSpray system and send control information using reliable protocol. Unfortunately, the last solution is very costly and obviously limits the scalability of the system.

Also, the implementation of the floor control poses a lot of problems in such environments. Usually it is achieved using tokens. The application that grabs the token gets the floor control. However, the problem occurs when the application that held the token terminates or loses contact with other applications. In such situation the new owner of the token must be elected.

Another important issue in the distributed environment is the session establishment. The clients that constitute the system do not have fixed time and place of operation. They may be run on any machine at arbitrary time. As a result, the new user that wants to enter collaborative session cannot determine any entry points to the current sessions without any additional mechanisms. There are several solutions to this problem. The system may provide session invitation mechanisms i.e. participants of the conference have the initiative to introduce new participants. The second solution is to provide session directories that contain information about all sessions and provide sufficient data to join them. The user that wants to create collaborative session puts information about that to the directory along with the timing and address. All users that want to take part in the session may use the information contained in the directory and connect to the session at specified time using specified address.

3.3.2 MESSAGE ROUTING

Messages in peer-to-peer architecture are distributed by the multicast service. This service may be implemented using unicast protocols such as UDP and TCP or using IP multicast protocols [18]. The former solution requires sending copy of the message to each receiver application. The sender must have information about location of all session participants. The applications that use IP multicast must know only the address of the specific multicast group. All messages that are sent for such address will be received by the applications that listen to it. Moreover, there is no need to send the copy of the message for each receiving user. In this way the application needs much less information in order to start

collaboration. Such solution provides efficient utilization of the network resources and ensures scalability of the system.

3.3.3 EXAMPLE: OPEN DVE

OpenDVE [16], product of the InSoft Incorporated, is an environment for the creation of the collaborative systems. The environment provides mechanisms for the session management, event distribution and data distribution. OpenDVE system has fully distributed architecture. Each user runs an application called Conference Engine that provides the communication with the Conference Engines of the other users. Collaborative applications, called Plugins, that may be developed and added to the system using provided API libraries communicate with the Conference Engine using inter process communication mechanisms. The application connected to the same Conference Engine may communicate among themselves as well as with the applications connected to the Conference Engines of the other users. In order to work together the users must establish a conference. The user may participate only at one conference at any given time. The Conference Engines run by the participants of the conference communicate one with each other providing communication framework for message distribution. The applications may use Conference Engine to pass their events. They can also use external channels for sending data and use OpenDVE infrastructure only for sending control information. One of the applications connected to the Conference Engine is dedicated to the conference management and receives the updates of the system state from the Conference Engine.

Each application supported by the OpenDVE system must perform register operation in the Conference Engine providing the specific Plugin identification number. The Plugin

identification numbers are use for determining the destination of the received message. Each message, sent by the particular application, is marked by the specific number so when the Conference Engine receives such message it may be passed to the proper application. Each application should implement its own communication protocol.

It is assumed that the Conference Engine runs all the time on user's machine. The system enables the users to establish the conference using the invitation mechanism. Whenever the user wants to invite another participant to the conference, he or she provides the system with the username and hostname of the participant or the list of participants. The Conference Engine establishes connection with the Conference Engine on the other machine and sends an invitation event. The response of the user is reported to the Conference Engine and to the conference manager Plugin of the user that initiated invitation procedure. If the asked user agrees to joint the conference the collaboration session is established. OpenDVE provides the tools for searching the Conference Managers of the other users on the local network. Such functionality is used for construction of the list of the potential conference participants.

Once the conference is established, the collaborative applications are started. The events sent by the applications are distributed to the peer applications of all conference participants. The Conference Engine sends information about the change in the local state of the system to other Conference Engines that take part in the conference. This information is sent in the form of the control events. Control events report such actions as joining the conference, leaving the conference, invitation and response to the invitation.

The OpenDVE environment uses reliable – TCP protocol for the basic communication. Data streams sent by applications may be transmitted using other protocols such as UDP and IP multicast.

3.4 OTHER ARCHITECTURES

Several existing systems are based on the architecture that cannot be assigned to any of the enumerated earlier types. For example, the InVerse system, developed by IBM, proposes a hybrid that employs client-server and peer-to-peer architectures. This collaborative system can handle various types of data streams e.g. real-time and transactional data. The system is composed of the server and the clients. The clients have capability of sending data directly to other clients. Data may be transmitted using UDP, TCP and multicast protocols.

Each client maintains reliable TCP/IP connection with the server. It is used for exchanging control information necessary for maintaining the state of the collaborative framework e.g. registration, queries for state information, group management, notification about state change use the reliable link. This connection is used also for transmission of reliable data e.g. text stream for the chat application.

The InVerse system uses notion of group as a set of the receivers of the particular piece of data. The system supports different types of groups that enable sending the data to different sets of the users e.g. to one user, to private groups, to public groups and to all registered users.

The data may be distributed using various types of channels. Choice of the channel depends on the current communication conditions of the transmitter and the receivers e.g.

support for the IP multicast. Depending on those conditions three communication channels are supported:

- client-server. The client sends the data to the server and the server transmits it to the group of receivers.
- peer-to-peer. The client transmits the data directly to all the clients in the group, IP multicast protocol is used
- hybrid mode. The part of the data is sent directly by the client to the receivers, the rest of the data is sent to the server and forwarded to the other group members.

In effect, the InVerse system employs different approaches to the management of the collaborative framework state and to the data distribution. For the maintaining the state of the collaborative framework strictly centralized approach is taken. The server keeps the information about the groups and makes this data accessible to the clients. The data distribution is supported by means of various types of channels. In this way, InVerse avoids problems with maintaining consistency of the system state and with scalability of the data distribution.

4 TANGO2 SYSTEM

This section presents general description of the Tango2 system. It also presents state management and message routing mechanisms implemented by the system.

4.1 OVERVIEW

4.1.1 PURPOSE

Tango2 system is a framework for building collaborative environments on the Web. It is being created based on the experience gained during the design and development of the Tango system. It concentrates on synchronous collaboration but it enables implementation of other models. It provides general collaboration mechanisms that may be employed by the developers using the Tano2 API. Tango2 offers environment that may be easily customized for various needs. It is platform independent - fully developed in Java. The set of supported application may be expanded easily without necessity of downloading – they are retrieved from the HTTP servers. The access to the environment is provided by the Web browser.

4.1.2 FUNCTIONALITY

Collaborative Features

The system supports following collaboration features:

- communities
- rooms
- session management
- event passing
- shared variables and locks
- session logging

In the following sections we will investigate their significance in creating collaborative environments.

Community

The communities are the virtual meeting places where the users may gather and perform various activities. Tango2 offers possibility to create such environments. They may be used for entertainment, providing on-line services, distant learning sessions as well as scientific collaboration. All those areas of activities involve collaboration and may be supported by Tango2 communities. From the system's point of view a community is a group of interconnected rooms that have uniform graphical interface. Within the boundaries of one community, a user may 'walk' from one room to another without restrictions. The rooms provide access to the related collaborative tools and gather people with similar interests and purposes. For example: virtual school community may have different rooms dedicated to the teaching different subjects with different tools. In the math room, the applet that shows function graphs to the students may be placed and biology room may contain applet that shows breathing process of the frog. Access to the community may be restricted using password checking procedure. This procedure allows construction of the public and group owned communities. It also enables building the commercial sites where user is charged for using provided services. The structure and content of the community are defined by the configuration files stored on the HTTP server and are interpreted by Tango2 demon. The community is identified by the URL where its configuration files are stored.

Room

Another important notion supported by Tango2 system is a room. A room is a part of the community. It groups the users that want to perform collaborative activities together. A

user may be only in one room at any given time using the same UID. A room contains various applications that may be used for collaboration. Access to the room may be controlled by one of the users, which decides who should be allowed to enter. The structure and content of the room, as of the community, is stored in configuration files that are accessible by HTTP server. Dynamic state of the room is stored on the Tango2 server. One room may be supported by only one server but one server may support several rooms. In this way, some degree of scalability is achieved. The rooms with small number of users may be placed together on one server and distinct servers may service the rooms with large number of users. Several servers may service one community.

Session Management

Session management tools provide the user with mechanisms that allow to initiate, terminate and control collaboration with the other users. The session management tools support following actions:

- session start and end
- session join and leave
- session invitation
- session state distribution

The Tango2 system uses notion of application type for distinguishing applications that use the same protocol for communication. The collaborative session may contain only applications of the same type. Thus, the user that wants to use several tools for collaboration must establish several sessions.

Tango2 system supports two types of sessions:

- predefined sessions
- dynamic sessions

Predefined session are used for room configuration and start automatically whenever the user enters the room. Those sessions may be different for different rooms. The dynamic session may be initiated by the user if he or she wants to extend array of used collaborative tools. The user has full control over dynamic sessions.

Tango2 system supports various models of the session depending on the needs of the constructed environment. Control locks determine the model of the session. Using the lock mechanisms it is possible to assign different roles to the different users. In this way, various session types may be implemented: from strictly asymmetric where only one person controls the session and sends data to the fully symmetric sessions. The asymmetric sessions are useful in distant learning applications when one user - the teacher controls the virtual lecture and interaction among other users should be limited. On the other hand in collaboration in the group of scientists everyone should have the same status in the session. Tango2 provides mechanism to address such situations. It provides an API that makes possible to embed the session management functions in the collaborative application. The resulting application is called a control application because it allows user to perform the session control operations.

Event Passing

Many collaborative applications are built using two complementary mechanisms:

- event passing

- shared data

Shared data information defines the content that the user works on. The events describe the actions of the user performed on the data. For example, the shared image browser may have capability of presenting the same pictures in different way e.g. normal and zoomed view, in such case the specific picture is the shared object and changes of the view are the application. The Tango2 system supports both mechanism in form of event messages used for event passing and shared locks and variables used for storing shared data.

Tango2 supports various modes of event passing. The event messages may be sent to all applications in the session or only to one of them. It is also possible to control rights to emit the events and assign roles of the event sources and the event receivers to the applications. The events may have arbitrary format because the system does not interpret event messages. The order of the messages is maintained and reliability the communication channels is provided.

Tango2 system provides API functions that enable application to send and receive event messages. It is part of Tango2 application API.

Shared Variables and Locks

Shared variables and locks is a main mechanism for application shared state implementation. Shared variables and locks may be created arbitrarily by the collaborative applications. Shared variables are used to store shared data and shared locks may be employed as a synchonization mechanism. Access to the shared variables and locks is controlled by the locking mechanisms that provides three modes:

- exclusive
- shared
- neutral

Some operations on the variables are permitted only in the specific modes described in Section 4.2 e.g. an application may modify the value of the shared variable if it locked the variable in the exclusive mode. The applications may use several variables and locks at the same time. Possibility of the application deadlock is eliminated by the order requirement in the variable locking. All the variables and locks are stored by the server. This approach assures consistency of the application shared state implemented using those mechanisms. Tango2 system supports different types of scope of the variables and locks as well as different type of owners (see Section 4.2). The scope defines visibility of the variable in the domain of applications. Using scope parameter it is possible to create shared data accessible by applications of different type.

The application may store data in shared variables in arbitrary format. The system does not interpret the value of the variables. It uses serialization mechanism to convert the data to the storage format. Change of the variable or the lock state is reported to all interested applications using notification mechanism.

Large flexibility in defining shared variables and locks parameters allows to implement complex patterns of collaboration. Floor control mechanism may be constructed using shared locks. Also, it is easy to accommodate users that joined session that started long time before. Initial state of the session may be maintained in the shared variable and transferred

to the new application. In this way a new user will see what happened before he or she joined the session.

Session Recording

Session logging mechanism provides support for the asynchronous collaboration in the Tango2 system. It enables users to record their collaborative sessions and replay them later. The content of the session is stored in the database where it accessible for search operations. The recording may be selective and concern only part of the activities performed during the session.

Other Functions

Tango2 system provides some other functions that do not support collaboration directly but make the environment management and maintenance easier:

- Tango2 system administration and control
- access control
- failure detection

Tango2 System Administration and Control

Tango2 system is controlled by the set of configuration files that define the structure of the communities and rooms. It also provides administration interface to the Tango2 server. The Tango2 server is one of the main components of the Tango2 system (see Section 4.1.3). It controls state management and message routing. The system administrator may directly view the state of the system and modify it dynamically using the interface e.g. user or session may be removed. Administration interface is accessible as a service run by the server on the

specific port. It uses text-based protocol, which offers the set of command for data browsing and modification. It also enables administrator to view the log of the system maintained by the server.

Access Control

Access control provides tools for limiting access to the communities. For each community the list of users with passwords is stored. Whenever the user wants to enter the community he or she will be asked for password. Using this approach various types of communities may be implemented - public, group owned, commercial etc.

Failure Detection

The system that works in the Web environment must be prepared for coping with various problems that occur in this environment. Changing connection parameters, different characteristics of the platforms where the clients run and large distribution of the system create potential danger of the errors in the system operation. Failure detection functionality enables keeping the system in the consistent state in spite of such problems. The Tango2 system provides support for hierarchical failure detection, namely the system tries to detect erroneous behavior of the collaborative tools as well as the crashes in the communication structure. When the system detects, using timeout mechanism, that the user is cut off from the collaborative framework it informs the other participant about this fact and removes the user from the system.

4.1.3 ARCHITECTURE

The Tango2 system is composed of servers, demons and collaborative applications. Figure 6 shows simplified structure of the system.

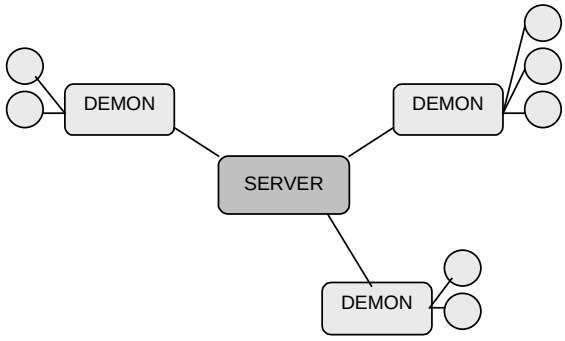

Figure 6: Architecture of the Tango2 system

Tango2 server is a central element of the Tango2 system. It may provide services for several rooms. This component manages the system state and distributes it among the demons. It handles session management, event passing, shared variables and locks. The server provides channels of communication for data message flow in the system. It implements failure detection and tracks irregularities in the behavior of the demons e.g. due to poor quality of the connections. The server may be configured to reflect the needs of the environment and it may be controlled by the administration interface. The Tango2 server communicates directly with the Tango2 demons using TCP/IP socket connections. It is implemented as a multithreaded Java application.

Tango2 demon is an intermediate element that enables communication between the applications and the server. It routes the messages from the server to the recipient applications and from applications to the server. It also provides access control. Tango2 demon interprets room configuration files and, using these data, it constructs user interface to the system. It also controls user transition from one room to another. It starts applications and terminates them. Tango2 demon interprets part of the control messages. It is important element in system state distribution. The demon maintains connections with all

collaborative applications started by the user and communicates with them using application API methods. It detects crashed applications - unfortunately this happens often to the applets - and reports such events to the server. It resides on the client side and is implemented as a Java applet.

Collaborative application is an application that implements Tango2 API and is able to communicate with the Tango2 framework. The applications may be of various types:

- Java applets
- Java script applications
- applications in languages that provide means of network communication

Collaborative applications define user interface to the system and serve as the collaborative tools. They may provide various functionality. They may be customized to the specific room character and role. Some of them have embedded session management functions and they fulfill important role of control applications. They enable user to create, terminate, join or leave sessions.

4.2 BASIC CONCEPTS

This section describes several terms that are important for understanding the state management and message routing issues in Tango2 system that are addressed in the remainder of the paper. Some of them were already mentioned in Section 4.1.2 but they are described here in greater detail and more formally.

Room is an abstract construct that separates sets of Tango users. Each room contains related applications that may be used by users for collaboration. Rooms may be customized

to reflect different interests of users. Rooms may be linked and the set of linked rooms defines community. Users see only users in the same room. Predefined sessions may be defined in the room, so that when the user enters the room collaborative applications start automatically.

Session is a logical group of applications working in a collaborative way. All applications in a session must belong to the same application group.

Application Type is defined by application group and application subgroup. All applications in the same group can communicate using the same protocol. Two applications that are in the same application group and subgroup are assumed to be the instances of the same application.

Shared Variable is used to store the shared state of the group of the applications. The name and the scope qualifier identify shared variable. A scope defines a domain where the variable is visible. Variables that have different scopes may have the same names. Shared variable has also an owner. The owner parameter defines lifetime of the variable. When the owner ceases to exist, the variable is removed. Table 1 shows allowed combinations of scope and owner parameters and how those parameters affect the lifetime of the variable.

Scope	Owner	Moment of Deleting
Session	Session	the session ends
Session	User	the user leaves the session
Room	Session	the session ends
Room	User	the user leaves the room
Server	Session	the session ends
Server	User	the user logs out from the

Table 1

Locking mechanism controls access to the shared variables. Variable may be locked for shared or exclusive access and be subject of the following operations:

- **create** - creates new variable.
- **read** - reads the value of the variable, may be executed only when the variable is locked in shared or exclusive mode by the application that performs reading.
- **write** - writes to the variable, may be executed only when the variable is locked in exclusive mode by the operation that performs writing.
- **change lock value** - changes lock value of the variable, variables must be locked by an application in alphabetic order.

Shared Lock is similar to the shared variable. It may be seen as a variable without value. Only the lock mode of the lock is important. Locks in Tango2 serve as a mechanism for ensuring synchronization among applications.

Polling Mechanism is used for detection of the crashed applications and demons. Unfortunately, web environment is highly unreliable and application crashes occur

frequently. This mechanism helps to determine if the application or demon are still active and eliminates not responding elements of the system. It improves state consistency. Polling mechanism in Tango2 system is hierarchical. The servers check the state of the demons and the demons check the state of the applications.

Operation is an protocol procedure that needs more complex communication that the request-answer exchange. Tango2 system protocol defines several operations:

- application start operation
- application terminate operation
- room enter operation
- change lock mode operation

Other protocol procedures may be handled in a simple request-answer pattern e.g. request for information about the room state is handled immediately and the message with the room state is sent. The execution of the operation follows more complex pattern. Usually it involves interaction of the several users. For detailed description of an operation example see Section 5.2.2.

Executing each of those operations may involve exchange of several protocol messages and interaction of several users (see dynamic model section for an example). All operations are managed by Tango2 server. It stores current state of each operation and controls validity of the transitions.

4.3 STATE MANAGEMENT

Tango2 system has complex state that is composed of large number of components. It is mainly maintained by the Tango2 server. The Tango2 server controls session management, room access, shared locks and variables, failure detection. Tango2 demons control room enter procedure and detects crashed applications.

4.3.1 COMPONENTS OF THE SYSTEM STATE

Demon State

Tango2 demon is a client part of the Tango2 system. Each demon maintains a copy of the room state in which the user currently is present. This state is transferred by the server when the demon connects to the system. From now on, the demon receives state updates whenever state of the room changes. State stored by the demon is accessible to the collaborative application. They may access the state data using API methods, however they cannot change the local state.

Apart from maintaining the copy of the room state, the Tango2 demon controls the condition of all running collaborative applications. It can detect crashed applications and report such situation to the Tango2 server.

Server State

State of the server contains the state of the rooms supported by the server, the state of the collaborative applications run by the users and other data that define server application settings:

- thread pool size - defines how many users may connect to the server, because each user is served by the separate thread
- buffer sizes - defines capacity of server output buffers
- timeout - defines how long the server waits for response from the users in polling routine
- variable maximum size - defines maximum size of the shared variable
- operation handlers - state of the protocol operations

Actions that change the state of the room will be described in greater detail in Section 4.3.2.

Room State

Room is a self-contained entity and its state is independent of the states of other rooms[1]. State of the room is composed of the following elements:

- room name. Uniquely determines the room supported by the server. The room with the same name but supported by the other server is different instance of the room. The room name does not change throughout the existence period of the room.
- active users. Server keeps list of the users present in the room. List of the active users changes as the users enter and leave the room. The server also stores the state of the users present in the room.

- state of the sessions. The server keeps list of the active sessions in the room along with their state.
- state of the applications. The server stores information about applications run by the users.

Information about the room state modification is broadcast to all users present in the room with one exception - change of the shared variables and locks with session scope is broadcast only to the participants of the session.

Session State

Session obtains the state in the moment of creation and its state is deleted when the session is terminated. Session state consists of following elements:

- session ID. Uniquely identifies session in the server. It is assigned by the server when the session is created.
- predefined mode. Tells whether session is predefined or dynamic.
- session name. Mandatory for predefined sessions and it is optional for dynamic session.
- session type. Determines kind application group of the applications that are run in the session.
- list of user in the session. Contains participants of the session.
- session control locks. Define model of the session.

[1] With the exception of the variables of the server scope

Session ID, name, mode and type do not change throughout the session existence. The list of the users changes dynamically as a result of following actions:

- enter/leave the session - list of the users is updated accordingly
- leave the room - whenever user leaves the room, he or she is also removed from the lists of participants of the sessions.

When the last user leaves the session, the session is removed and its state is deleted.

Each session has its own set of control locks. Three types of the session control locks exist:

- JOIN lock
- TERMINATE lock
- MASTER lock

Using of JOIN lock access to the session may be limited. When this lock is exclusively locked by one of the users, all request of joining the session must be approved by the owner of this lock. TERMINATE lock has affects the terminate operation. If the user wants to terminate application of the other user in the session, he or she must acquire the TERMINATE lock in shared or exclusive mode. If the destination user did not acquire the TERMINATE lock in the shared mode, the application is terminated without asking him of approval, otherwise he or she will be asked for acceptance. User that gets MASTER lock in exclusive mode is the only one who may send events in the session. The mode of the lock may be modified according to the procedure described in Section 4.3.2.

User State

State of the user consists of following elements:

- user ID and user name. Uniquely identify user in the system. User ID is assigned when the user connects to the server. No two users with the same ID or name are allowed on the same server. User name is declared by the user during login procedure. User ID is used as a key in all operations.
- room enter time. Specifies when the user entered the room.
- session enter time list. Contains session enter time of the user for each session he or she participates in. This list is modified whenever user enters or leaves the session.

Application State

Control state of the application consists of following elements:

- application group. Specifies the set of the applications that may communicate using the same protocol and to which the application belongs.
- application subgroup. Indicates specific implementation of the application. Application group and subgroup are assigned to the application in the configuration file.
- access key. Identifies uniquely an application in the server. It is used for identification in access to the shared variables and locks. Access key is

assigned to the application by the server in the moment of the start and does not change throughout lifecycle of the application.

- variables and locks state. State shared with other applications.

State of an application in the Tango2 system may be of two types: application local state, kept by an application and shared state managed by Tango2 system. It is up to implementation how to use those two approaches. Some applications will rely only on local state and send events to other applications whenever state changes, some will use shared state model and some will use both of them.

Application may allocate (create) new variable and since this moment the variable becomes component of the shared state of all applications in the session, in the room or in the server depending on the scope of the variable. Modification of the shared state of the applications in the session is performed by modification of the state of the shared variables accessible to the application. State of the shared variable will be described in detail in the following sections.

Shared locks and variables have important role in application state management. Applications may use locks and variables for communication and synchronization. This is the main support for synchronous collaboration provided by Tango2 system.

Shared Lock State

Lock state consists of following components:

- name. Uniquely identifies the lock within the specified scope.

- scope. Composed of the scope type: server, room or session and scope identifier : room name or session number. Name and scope define uniquely lock in the server.
- owner. Defines lifetime of the lock.
- ask/ans mode. Defines whether application that exclusively locked the lock is asked when the other application tries to acquire the lock.
- lock mode. Defines the condition of the lock.
- list of access keys. List of the access key of the applications that currently hold the lock.

The name, the scope, the owner and the ask/ans mode do not change through lifetime of the lock. They are specified by the application in the creation request.

Another element of the lock state is the lock mode that defines the way the lock is used:

- exclusive - only one application holds the lock
- shared - several applications hold the lock
- neutral - no application holds the lock

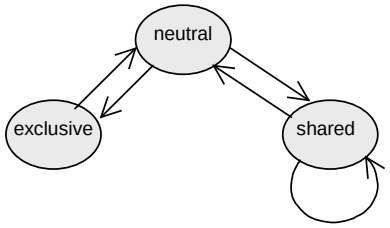

Figure 7: Allowed transitions of the lock mode

In order to change lock mode, application must send request specifying the lock it desires to hold along with the lock mode. The possible transitions of lock modes are illustrated in Figure 7. In order to acquire lock in the shared mode, lock must be in neutral mode or shared mode. In order to acquire lock in the exclusive mode, lock must be in neutral mode. The server controls access to the lock by using access keys that are assigned to applications.

Shared Variable State

Shared variable state is similar to the shared lock state. It contains all elements of the locks state. New component is a value of the variable. The value of the variable is a component of the shared content that may be accessed by the collaborative applications.

Value of the share variable may be accessed by applications by read and write operations. Those operations may be performed only in the specific lock mode of the variable. Following rules apply:

- read may be executed only in shared and exclusive lock modes
- write may be executed only in exclusive lock mode

Application that creates variable may choose initial lock mode for the variable and initialize it with desired value.

Operation State

Operation is created when the server receives the message that initiates the operation. Operation state consists of following generic elements:

- operation type. Depending on the request, the operation of the specific type is created. There are four types of the operations:
  - room enter. Handles procedure of entering the room.
  - application start. Handles the procedure of the application starting.
  - application terminate. Handles the procedure of the application termination.
  - variable/lock change lock mode. Handles the procedure of changing variable or lock mode.
- operation ID. Identifies uniquely operation in the server.
- ID of the initiator. Defines the ID of the user that initiated the operation. After the operation is finished, the result of the operation is reported to this user.
- stage. Defines the stage of the operation. Each operation has several stages specific to its type. In the process of message exchange operation transits from one stage to another. When the operation reaches the final stage, status

of the operation is reported to the operation initiator and operation terminates.

Along with the generic components of the operation state, information specific to the type of operation is stored. As an example we will investigate in detail the state data of the application start operation:

Application Start Operation

As a result of the application start operation a new application is started for the user. Specific state information for this operation contain:

- target user ID. Determines the user for whom the application will be started.
- session ID. Determines in which session application will be started.
- session name. Used if the application is started in the named session. Determines the session where the new application will be started.
- application group and subgroup. Identify the program that should be started.

Application start operation has several specific stages:

- INIT. The operation is initialized.
- START_ANS_WAIT. The operation waits for the answer of the target user if he or she agrees for application start.
- JOIN_ANS_WAIT. The operation waits for the answer of the user that holds JOIN lock of the session if he or she approves start of the new application.

- START_NOT_WAIT. The operation waits for the notification about application start (successful or failed).

Some of those stages may be not used in some cases e.g. when nobody holds the JOIN lock, the JOIN_ANS_WAIT stage will be skipped. The state diagram for this operation is described in the Section 5.2.2.

Execution of any operation may result in the change of the system state e.g. application start operation will modify state of the session to which application was added. Modification of the state resulting from the operation execution is delayed as long as possible so that rollback may be performed if the failure occurs.

4.3.2 STATE MODIFICATION

The state of the system is modified as a result of various actions executed by the actors in the system. Three types of actors exist: the users, the system administrator and the system itself. Each of those actors performs different actions that affect the system state. This section describes what changes of the state may occur and what causes them.

User Actions

Users modify the state of the system by their actions in the collaborative environment. In Tango2 system those actions may be divided into two groups:

- room level actions
- session level actions

Room Level Actions

Room level actions enable user to enter or leave the room where the user may communicate with the other system participants. Access to the room is controlled by the ENTER lock that has room scope. Room lock - ENTER- enables implementation of the access policy. Person that exclusively keeps the ENTER locks has right to deny access to the user that wants to enter the room. Ownership of the lock may change according to the locking protocol described in next sessions. Following sections specify change of the system state as a result of the room level actions.

room enter

When a new user enters the room the Tango2 server creates a structure that describes user's state. During this process, the server assigns the unique ID to the user that is treated as a key in all operations. Also, room enter time is noted and stored in the user state data. Finally, the user state is added to the list of users in the room and information about the room state change is broadcast to all users in the room. Optionally information about new user may be recorded in the server log.

room leave

When the user leaves the room, all structures that describe the state of the user are deleted. State of all applications that were run by the user is also deleted. Information about room state change is broadcast to all users in the room.

room change

Room change is composed of room leave and room enter actions.

Session Level Actions

Session level actions allow user to collaborate with the other users namely they enable to: create, join, leave and terminate sessions as well as start and terminate applications for other users.

session create

A new session is created when the user starts an application that does not participate in any existing session. In the first step new session ID is created along with the structures that describe session state. Fields of the session state structures are set to given session name and type. The system creates control locks for the session. Another step involves adding the stat of the new application to the state of the session. This stage is described in the next section.

session join

User joins the session when he or she starts an application that is able to communicate with the applications executed by the participants of the session. In this process, new structure that represents the state of application is created by the system. Unique access key is generated and stored along with the application group and application subtype. Information about new user in the session is added to the session state structures and state of the user is updated with session enter time information. Finally, the server informs all users in the room about change of the state.

session leave

User leaves the session when the application that communicates with the applications of the other users in the session is terminated. The application is removed from the list of the

applications in the session and from the user's state session enter time for the abandoned session is removed. All variables and locks with the session scope visible in the session and owned by the leaving user are terminated.

session terminate

The session is terminated when the last user leaves the session. In such case, structures that hold session state are removed from the room state. All variables and locks owned by the terminated session are deleted.

application remote start

This action is similar to the session join action. This action is executed when one user wants to start the application for another user. State modification is the same as for session join.

application remote terminate

This action is similar to the session leave action. The action is executed when one user wants to terminate the application of the other user. State modification is the same as for session leave.

Application Actions

Application actions modify the shared state of the applications.

shared variable create and shared lock create

Shared variable create action creates a shared variable in the system. The structure that describes the state of the variable is created with fields name, scope, owner and ask/ans

mode initiated. The shared variable is created in the lock mode that is specified in the create request (exclusive, shared or neutral). The application that created shared variable has possibility to initialize the variable with the arbitrary value, which is stored along with other state data of the variable.

Shared lock create action is similar to the shared variable create action. The only difference is that the application cannot specify any value for the lock.

shared variable/lock lock mode change

Change of the lock mode modifies the state of the shared variable or lock. Information about lock mode changed is broadcast to all applications that are in the scope of the variable or lock.

shared variable read/write

Read action enables access to the value of the shared variable and does not change the state of the variable. Write action modifies the value of the variable with new data and as a consequence modifies the state of the shared variable. This change is reflected in the structures that represent the state of the variable.

event send

This action may be used to send shared application events. Depending on the handling procedure of the event it may change the state of the collaborative application. However, it does not change the state of the collaborative framework or the state of the shared variables and locks.

Server Actions

The server performs some initial actions that are necessary for proper configuration of the system. The information most important of all is the creation of the structures that hold the state of the rooms and maintaining the system and the access log.

room create

The server creates the rooms according to the information contained in the server configuration file. This specifies the list of the rooms to be supported by the server. Each room is represented by the URL of the file with the description of the room.

system log write

The server stores information about important events in the system log. This data is used mainly for debugging.

access log write

The server stores information about logged in users in the access log. Access log stores following data: user name of the user, entered room URL, login time, logout time, hostname, time of the session enter and time for the session leave for each session in which the user participated. This information is used for construction access statistics.

Administrator Actions

Administrator actions are executed by means of the administrator interface supported by the server. Using the text based protocol the administrator may browse and modify the state data.

data browsing:

The system administrator has access to the following information using administrator interface:

- server information. Parameters of the server: thread pool limit, buffer size, timeout, variable maximum size setting.
- room information. Names of the supported rooms with the number of users and session in the room.
- user information. User name and ID, hostname of the user, session enter times and room enter time.
- session information. IDs and usernames of the session participants
- variable and lock information. List of variables with the information that constitutes variable state.
- operation information. List of operations with their types and stages.
- access log content. List of accesses to the server.
- system log content. Information about events in the system.

data modification :

The system administrator may execute following actions:

user delete.

Removes the user from the room. Change of the room state is the same as for the room leave user action. All users in the room are informed about change of the room state.

session delete.

Closes the session in the room. Change of the state as in the same as for session terminate action. All users in the room are informed about change of the room state.

operation delete.

Terminates an operation and removes the structures that keep operation state. User that initiated the operation is informed about operation failure. Some operations in specific stages cannot be deleted e.g. application start operation when waiting for the notification that the application started.

server parameters set

This action enables setting of the thread pool threshold, buffer size, timeout, variable maximum size parameters. Modification of those setting affects the server state.

configuration file reread.

Forces the server to read the configuration file. The rooms that are supported by the server but are not enumerated in the configuration file are deleted. The rooms that are not supported currently by the server but found in the configuration file are created.

server shutdown.

This action terminates the server application and deletes dynamic state of the system.

Failure Detection

The Tango2 system tries to detect applications with erroneous behavior and not responding demons. If the application does not respond to the messages, the demon

executes application terminate action and removes application from the system. Another case occurs when the demon stops responding to the server messages. After specified timeout period, the user that was running demon is removed from the system as in the user delete action of the system administrator.

4.4 MESSAGE DISTRIBUTION

The message distribution mechanism is essential for the maintaining communication between distributed components of the system. This section describes the types of the messages that are handled by the system. It also examines the available channels of communication in the Tango2 system and their implementation.

4.4.1 TYPES OF MESSAGES

Control Messages

Control messages are used for communication between various components of the Tango2 framework. Tango2 control messages may be divided into several groups:

- room management - control process of entering and leaving the room as well as sending room state to the users that are new in the room
- session management - control session creation and termination, session joining and leaving
- state update - inform about system state change
- polling messages - detect crashed demons

Application Event Messages

Application event messages are used for communication between applications that are in the same session. Whenever an application wants to inform other applications in the session about any action it sends application event message.

Application Shared State Messages

This group of messages is used for shared variables and shared locks management. They are used for read, write and change lock operations. They also inform about change of the state of the variables or locks.

4.4.2 CHANNELS OF COMMUNICATION

The Tango2 system provides and uses various types of communication channels. They are used for inter-application communication and collaborative framework management. All channels of communications are managed by the Tango2 server which accepts and responds to the messages. Message sent by an application is passed to the Tango2 demon, which sends the message to the Tango2 server. Messages sent by the server are accepted by the Tango2 demon and interpreted or sent to the recipient application. Apart from the channels provided by the Tango2 system, applications may establish direct links of communication. Such solution is necessary for transmission of real-time data. The Tango2 system implements following channels of communication:

Server Broadcast

Sends message to all users that are connected to the server. The server repeats procedure of room broadcast for each room supported by the server with the same message. This channel is used for notification about change of value of the variables and locks that have server scope.

Room Broadcast

Sends message to all users that are present in the room. For determining recipients of the room broadcast, room state data managed by the server are used. This channel distributes notifications about room state changes and notifications about shared application state changes i.e. variables and locks that have room scope.

Session Broadcast

Sends message to all users that are in the same session. The server determines recipients of the session broadcast using session state data. This channel is used for application event passing to all applications in the same session. Session broadcast is also used for notification about shared application state changes i.e. variables and locks that have session scope.

Unicast

Sends message to the specific user. The connection with the specific user is uniquely determined by the name of the room where the user is present and the user ID maintained

by the server. This channel is used for sending control messages other than system state updates and sending event messages to the specific applications.

5 TANGO2 SERVER

The Tango2 server is the main component of the Tango2 system. It manages the system state and distributes the messages. This section describes the structure and design of this application. Detailed object and dynamic models of the several server modules are also presented.

5.1 TANGO2 SERVER ARCHITECTURE

5.1.1 GENERAL STRUCTURE

Tango server consists of several modules. Figure 8 illustrates general structure of the server and control relationship between the modules.

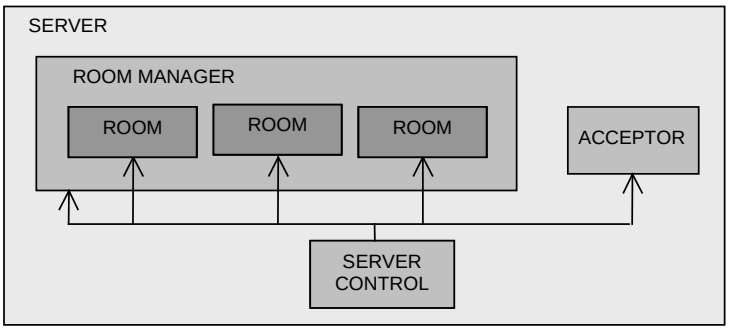

Figure 8: Tango2 server architecture

The main parts of the server are:

- Room Manager

- Acceptor
- Server Control

Room Manager and Rooms contained in this module as well as Acceptor are controlled by Server Control module. Acceptor and Room modules maintain connections with Tango2 demons. Server Control may be accessed by system administrator using administration interface.

Server Control is a module that controls operation of the whole server. It means that it is able to execute all operations enumerated in Section 4.3.2 on behalf of the system administrator, calling specific procedures implemented in Room Manager, Room and Acceptor modules. Server Control runs a service that allows the administrator to connect to the server and view or change the state of the server and the system. Only one such module exists in running server.

Acceptor is a module that accepts connections from the Tango2 demons. It is like a waiting room for users before entering specific room. When a user enters the room, structures that control the connection with that user - User module - are moved to the corresponding Room module. Only one such module exists in the running server.

Room Manager is a container that stores all Room modules and enables access to them. It also provides support for server broadcast and variables and locks of the server scope. Only one such module exists in running server.

Room module represents a room in Tango system. All data and functionality necessary for execution of operations that are specific to the room are concentrated here. There is no

communication between different rooms. The Room module maintains connections with all Tango2 demons that represent users present in the room. The number of Room modules in a running server is equal to the number of rooms declared in server configuration file.

5.1.2 ROOM STRUCTURE

Room module represents a room in the Tango system. Figure 9 shows all important sub-modules that are contained in Room module along with control and access relationships between them:

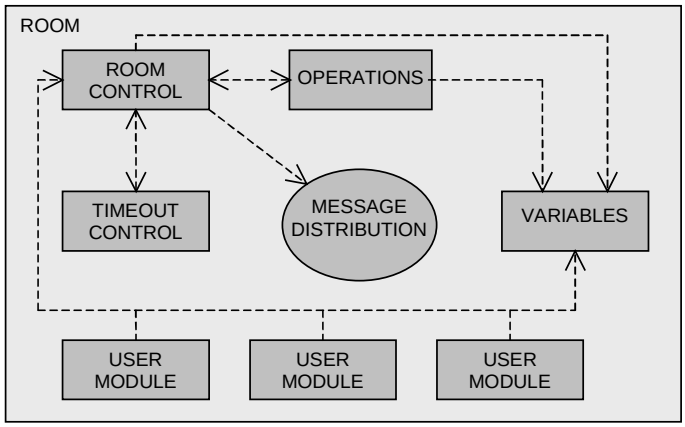

Figure 9: Architecture of a Room module

Dashed line denotes access and modification relationship e.g. user module may access information in Room Control module and update system state data. More detailed description of Room components and interactions between them may be found in the next sections.

Components

Room Control stores the room state data:

- sessions established in a room
- users present in the room
- applications run by the users
- predefined sessions

Room Control module also provides means for access and modification of this data. There is only one Room Control module in a Room module.

User Module communicates directly with one Tango2 demon. This module represents a user in the room. It means that whenever a user enters the room, corresponding User Module appears in the Room module. User Module implements part of the server - demon communication protocol. It handles directly all messages that do not involve sending queries to other Tango2 demons.

Operations module manages operations executed in the Tango system. This module controls the state of all operations and routes protocol messages to the proper operations. There is only one Operations module in a Room module.

Timeout Control tries to detect crashed demons. It sends messages to all demons in the room. If any demon does not respond for specified period, Timeout Control module assumes that a crash occurred and notifies Room Control module. There is only one Timeout Control module in a Room module.

Variables module stores variables and locks that are created by applications executed by the users present in the room and by the server. This module also provides means for access

and modification of those variables and locks. There is only one Variables module in a Room module.

Message Distribution module passes messages from the source module to the destination module or modules. It has several modes of message distribution available:

- broadcast to all User modules in the Room module
- broadcast to all User modules in the Room module that represent users with applications in the specified session
- passing of the message to the specified User, Operations or Timeout Control modules

Message Distribution module is used only for passing messages that are specified in the server-demon communication protocol. Communication between modules that does not involve passing mentioned messages is assured by other mechanisms. There is only one Message Distribution module in a Room module.

Dependencies

Apart from message exchange, components of the Room module maintain other links denoted by the dashed lines in Figure 9. Their purpose is to access or modify an information that is contained in different module. Following associations exist:

- **User module - Room Control.** User module accesses the data about users and sessions contained in Room Control module, it also notifies the Room

Control about modification of the room state data resulting from handling protocol messages.

- **Operations - Room Control.** Operations module accesses the room state data contained in the Room Control module necessary for protocol operations handling. Operations module also notifies Room Control about changes in the state of the room resulting from the execution of protocol operations. The Room Control module notifies Operations module about change of the state of the users e.g. crash of the demon.
- **User module – Variables.** User module accesses data about the shared variables and locks contained in the Variables module. User module also requests modification of the values of the variables and locks as a result of protocol message handling.
- **Operations – Variables.** Operations module accesses data about the shared variables and locks contained in the Variables module. Operations module also requests modification of the values of the variables and locks as a result of protocol message handling.
- **Timeout Control - Room Control.** Timeout Control module notifies the Room Control about crashed demons. Room Control module notifies the Timeout Control about changes in the users state.
- **Variables – Room Control.** Room Control notifies the Variables module about changes in the room state.

- **Room Control - Message Distribution.** Room Control notifies the Message Distribution module about changes in the room state.

5.1.3 FLOW OF MESSAGES

The message routing is one of the important tasks of the Tango2 server. In previous sections, the channels of communication supported by the Tango2 system were described. This section shows how the Tango2 server handles incoming and how it distributes outgoing messages.

Majority of the messages are handled and distributed in the boundaries of one Room module. Therefore we will focus on the Room module structure and describe how various components of this module interact to provide the required services.

Incoming Messages

Incoming messages may be handled in three different modules of the Room:

- **User module** handles part of the request messages and all application event messages.
- **Timeout Control** handles timeout control responses.
- **Operations** handles part of request messages, answers to the queries and all notification messages.

User module receives all incoming messages and decides whether they should be handled or forwarded to other modules. Messages to be forwarded are passed to the Message

Distribution module which delivers them directly to the destination as illustrated in Figure 10. Therefore following routes for incoming messages are available:

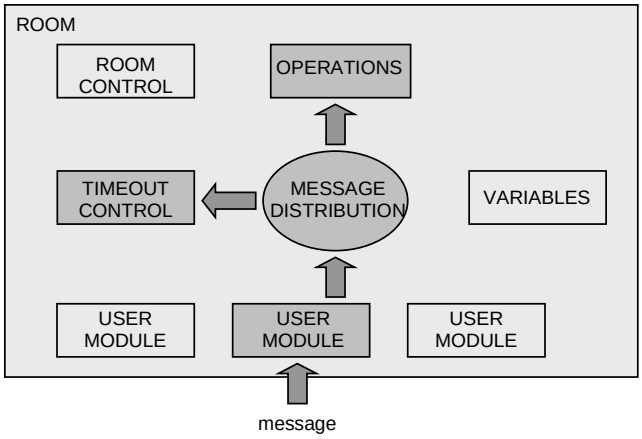

Figure 10: Routing of incoming messages

- User module
- User module – Message Distribution - Operations
- User module – Message Distribution – Timeout Control

Such message routes are forced by a fact that implementation of the server-demon communication protocol is distributed among several sub-modules of the Room module, namely User modules, Operations and Timeout Control. As a result the message must be transferred to the sub-module that knows how to handle it.

Outgoing Messages

Main component that supports the outgoing message routing is the Message Distributor module. Outgoing messages may be transmitted in four modes:

- server broadcast
- room broadcast
- session broadcast
- unicast

Following sections describe those modes:

server broadcast

This mode is used for notification about events that affect all users connected to the server. Room Manager module sends outgoing message to the Message Distributor module in each Room. The Message Distributor module executes room broadcast.

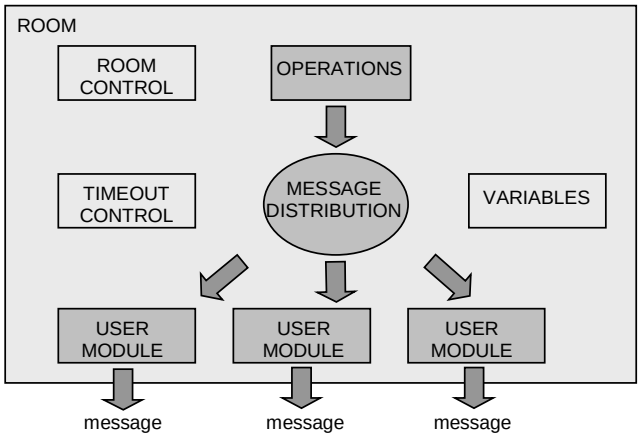

Figure 11: Room broadcast

room broadcast

Room broadcast is used to notify users about changes in the state of the room. The Operations or User module may initiate room broadcast. Figure 11 shows the message flow in the transmission initiated by the Operations module. Message to be transmitted is passed

to the Message Distributor module, which hands it to all User modules present in the room. Then, each User module sends the message to the destination.

session broadcast

Session broadcast is used to send shared applications events to all members of the session. Any User module in the Room may initiate this transmission. Figure 12 shows the message flow that occurs when session broadcast is executed.

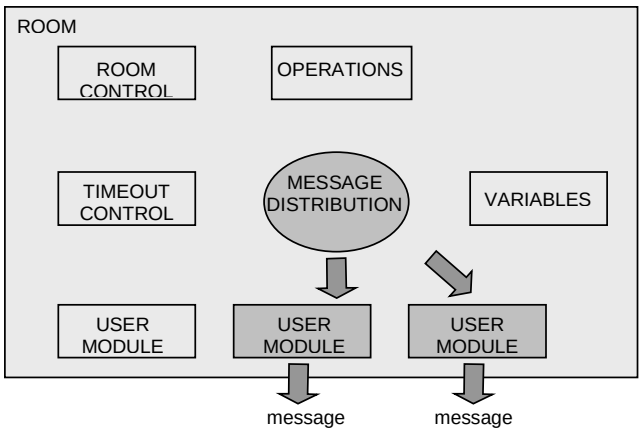

Figure 12: Session broadcast

User module that wants to send the event to other session participants passes the message to the Message Distribution module. The Message Distribution module passes the message to all User modules that represent participants in the session (event to the sender). The User modules send the message to the destination.

Unicast

Unicast is used to send a message to the specific user. All acknowledgement, action and query messages to the demons are sent using this channel. The Operations module, Timeout module or any User module in the Room may initiate sending a message in unicast mode.

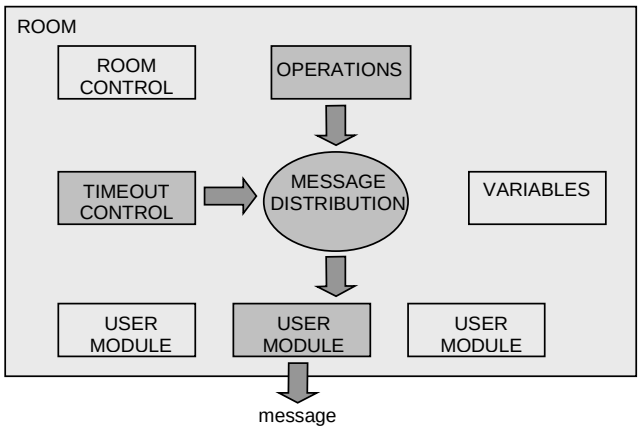

Figure 13: Unicast

Figure 13 shows message flow when the unicast is executed. The message created in the Operations or Timeout Control modules is passed to the Message Distribution module with the information specifying the receiver of the message. The Message Distribution module passes the message to the User module that maintains contact with the specified demon. The User module sends the message directly to the destination. Unicast may be also initiated by the User module. In such case, User module sends the message directly to the destination without assistance of the Message Distribution module.

5.2 TANGO2 SERVER – ANALYSIS AND DESIGN

This section contains object and dynamic models of several modules of the Tango2 server. We present only those aspects of the design that address state management and message routing issues. All presented models use the notation described in [19].

5.2.1 OBJECT MODEL

In this section we will discuss design techniques used in the Tango2 server design phase. We will also present object models of the modules of the server that are responsible for state management and massage passing in Tango2 system.

Interfaces

In design of the Tango2 server we tried to distinguish various functionality that the server should offer and assign those functionality to different modules of the system. For each such module we constructed set of interfaces that fully define services offered to the other modules. In this process we tried to follow indications presented in [20] in order to achieve maximum flexibility, make the system easier to implement and easier to change. Also, we used several patterns described in [20] that made the system structure more clear. In this section we will enumerate the most important groups of interfaces that are part of the object design of the server along with the short description. Following sections will show how those interfaces define functionality of the specific modules.

Interfaces used by server module may be divided into several groups:

Room management:

Provides access to the room pool, adding and removing rooms, access to the control modules of the specific room

Room control data management:

Provides access to the state data of the room about active sessions, users and applications. Defines functionality of the room and session state observers.

Variables and Locks Management:

Provides access to the shared variables and locks. Enables reading and writing to the specific variable, enables change of the lock mode of the shared lock or shared variable.

Operations Management:

Enables operations management, operation adding, execution and retrieving its status as well as operations error handling

Message Handling and Distribution:

Provides generic interface to all messages as well as the interface for the specific groups of messages like operation messages. Enables access to the type information of the message and provide methods for decoding and encoding. Other interfaces from this group support message distribution and configuration of the distributor modules. Finally, it provides access to the specific implementation of the message handlers.

Timeout Control:

Contains interfaces for Timeout module and for modules that are interested in the information about crashed demons (timeout data observers).

Demon-Server Connection Management:

This group provides generic interfaces for connection handling, message sending and connection closing. Defines functionality of the observer of the connection state, message receiving, connection error handling. It also defines interfaces for the threads that receive messages from the demon, the threads that send the messages to the demon as well as the error handling in both cases.

New Users Management:

Contains interfaces for handling new users and introducing them into a room.

Administration and Maintenance:

Defines interfaces to the specific commands that enable their command execution. Also defines administration interface to the room and enumerate administrator operations on the room. This group also defines the administrative operations on the specific modules.

Object Models of the Selected Server Modules

In this section we will examine object models of the selected modules and describe their structure.

Room Control Module

Each room in the Tango2 system has dedicated structures that manage its state. Room Control module contains main portion of the room state, namely sessions and users state as well as the part of the application state (access keys). Object models of the Room Control module are presented in Figure 14 and Figure 15.

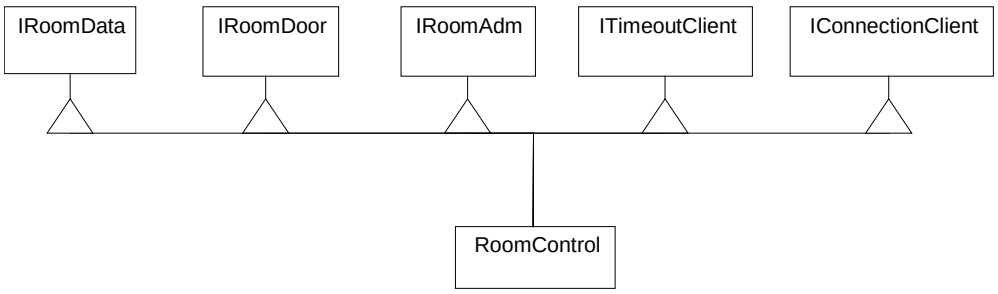

Figure 14:Part of the object model of the Room Control module

Figure 14 presents all interfaces supported by the Room Control module:

- IRoomData
- ITimeoutClient
- IConnectionClient
- IRoomDoor
- IRoomAdm

IRoomData provides access to the room state data and defines mechanism for state modification. Using this interface, other modules may add or delete users from the session, remove users from the room, check the status data of the sessions and users. It belongs to the Room Management group of interfaces.

Room Control module controls access to the room and decides about letting the user in. By using IRoomDoor interface, Acceptor module brings in the new user to the room. IRoomDoor belongs to the New Users Management group of interfaces.

Room Control module accepts notification from other modules whenever event that is important for the system state takes place. ITimeoutClient interface defines handlers for client crash error. Whenever the Timeout module detects the failure of the clients it uses ITimeoutClient interface to inform about that. IConnectionClient interface is used for socket connection related problems and is used by User module to notify about broken connections. ITimeoutClient belongs to the Timeout Control and IConnection Client belongs to the Demon-Server Connection Management group of interfaces.

Finally, IRoomAdm interface is used for actions defined in the administration interface that must be. It belongs to the Administration and Maintenance group of interfaces.

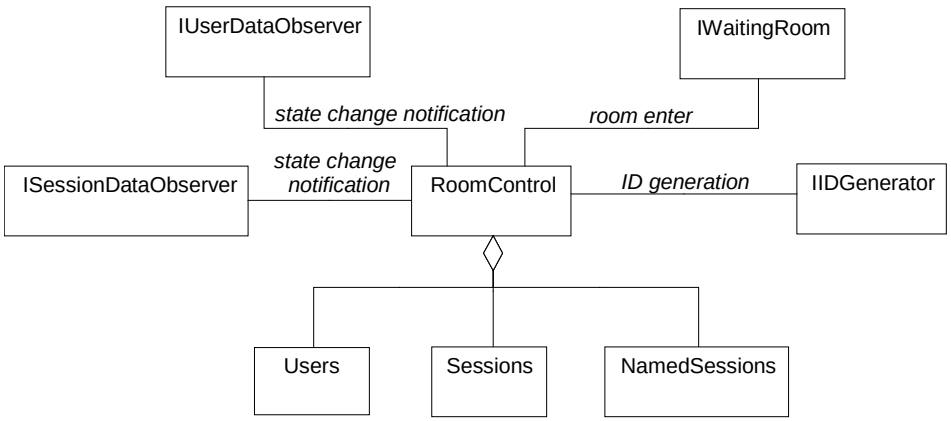

Figure 15: Part of the object model of the Room Control module

The Room Control module keeps state data in several submodules that manage specific information about sessions, users and named sessions as presented in Figure 15. Room Control module contains following submodules:

- Sessions
- Users
- NamedSessions

Sessions submodule keeps information about all sessions in the room. Additional information, specific to the named sessions is stored in the NamedSessions submodule. Those both structures keep all session state data defined in section 4.3.1. User submodules manages state of the users in the room.

RoomControl module communicates with other modules using following interfaces:

- IWaitingRoom
- ISessionDataObserver

- IUserDataObserver
- INumberGenerator

IWaitingRoom interface is used for communication with Acceptor module. Using functions from this interface, Room Control may accept new users to the room. It belongs to the New User Management group.

Several modules depend on notifications about room state change (Operations, Variables, Message Distributor). Those modules use ISessionDataObserver and IUserDataObserver interfaces to receive those notifications. Whenever the users state changes, Room Control notifies IUserDataObserver interface. ISessionDataObserver is used when the state of the sessions changes. Both interfaces belong to the Room Control Data Management group.

Room Control uses INumber interface to retrieve unique access key for the applications used later for shared locks and variables locking procedure. It belongs to the Room Control Data Management group.

Message Distribution Module

Message Distributor module is the part of the server that takes care of message routing. Its object model is presented in Figure 16.

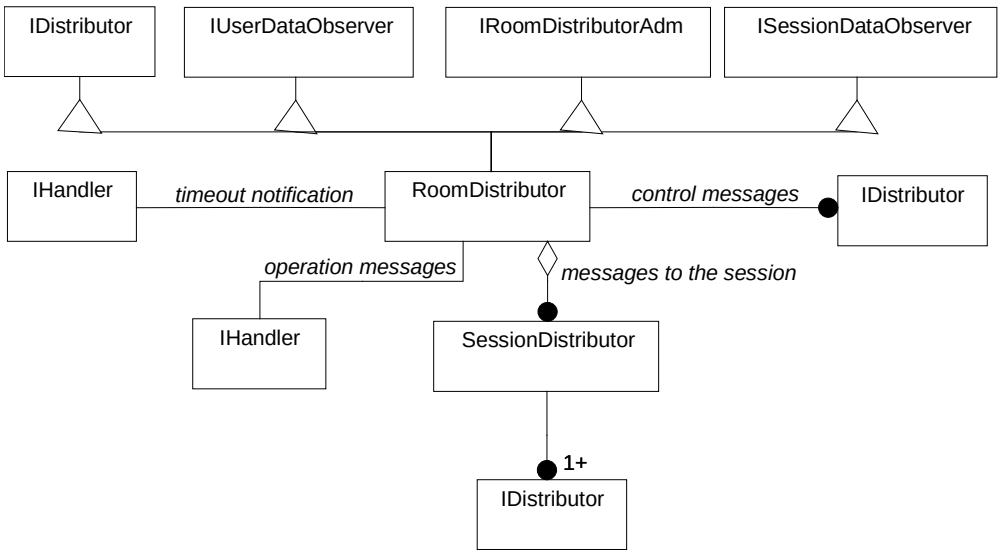

Figure 16: Object model of the Message Distributor module

Message Distributor module supports following interfaces:

- IDistributor
- IUserDataObserver
- ISessionDataObserver
- IRoomDistributorAdm

IDistributor interface defines channels of message distribution supported by Message Distributor module. It offers other modules possibility of:

- sending a message to the specific user
- sending a message to the Operations module
- sending a message to the Timeout module
- broadcast a message in the session
- broadcast a message in the room

- broadcast a message to all users connected to the server

IDistributor belongs to the Message Handling and Distribution group.

IUserDataObserver and ISessionDataObserver enable the Message Distribution module to receive notification about change in the room state and modify its structures accordingly e.g. when the user leaves the session it should not receive messages broadcast in this session any more. Both belong to the Room Control Data Management group.

IRoomDistributorAdm interface enables Server Administrator module to execute administration procedures on this module and retrieve status data. It belongs to the Administration and Maintenance group.

Message Distributor module contains SessionDistributor submodule that manages message broadcast among the participants of the session. Message Distributor module also communicates with following interfaces:

- IHandler
- IDistributor

Every module that wants to receive messages implements IHandler interface. Message Distributor has direct links to the Operation and Timeout modules that use this interface. It sends outgoing messages to the User modules using IDistributor interface. Both interfaces belong to the Message Handling and Distribution group.

5.2.2 DYNAMIC MODEL

In this section we will present the dynamic model of the ExecStartOperation class that implements session enter operation. This is one of the classes that implement protocol of the server – demon communication. It is part of the Operations module. Figure 17 shows how the state of the system changes as a result of the low level operations (message arrivals). In the figure RUID denotes the ID of the user that requested the session enter operation, DUID denotes the ID of the user that is the target of the operation i.e. who will join the session. JOIN_ANS_WAIT, START_ANS_WAIT and START_NOT_WAIT denote the stages of the operation described in section 4.3.1.

The operation is initiated on the requests of the user that wants to enter or invite somebody else to the session. The handler of the operation tries to obtain the consent of the user that keeps the JOIN lock (JOIN_ASK message) and the target user (START_ASK message). Then collaborative application is started for the target user (START_ASK message) and the status of the operation is sent to the user that requested the operation (START_ACK message).

The diagram presented in Figure 17 is simplified. It does not take into account the failures of the various actions as: receiving the corrupted message, no response from the asked user, termination of the connection with the user that is needed for the completion of the operation. Such events usually result in the termination of the operation and sending appropriate message to the initiator of the operation.

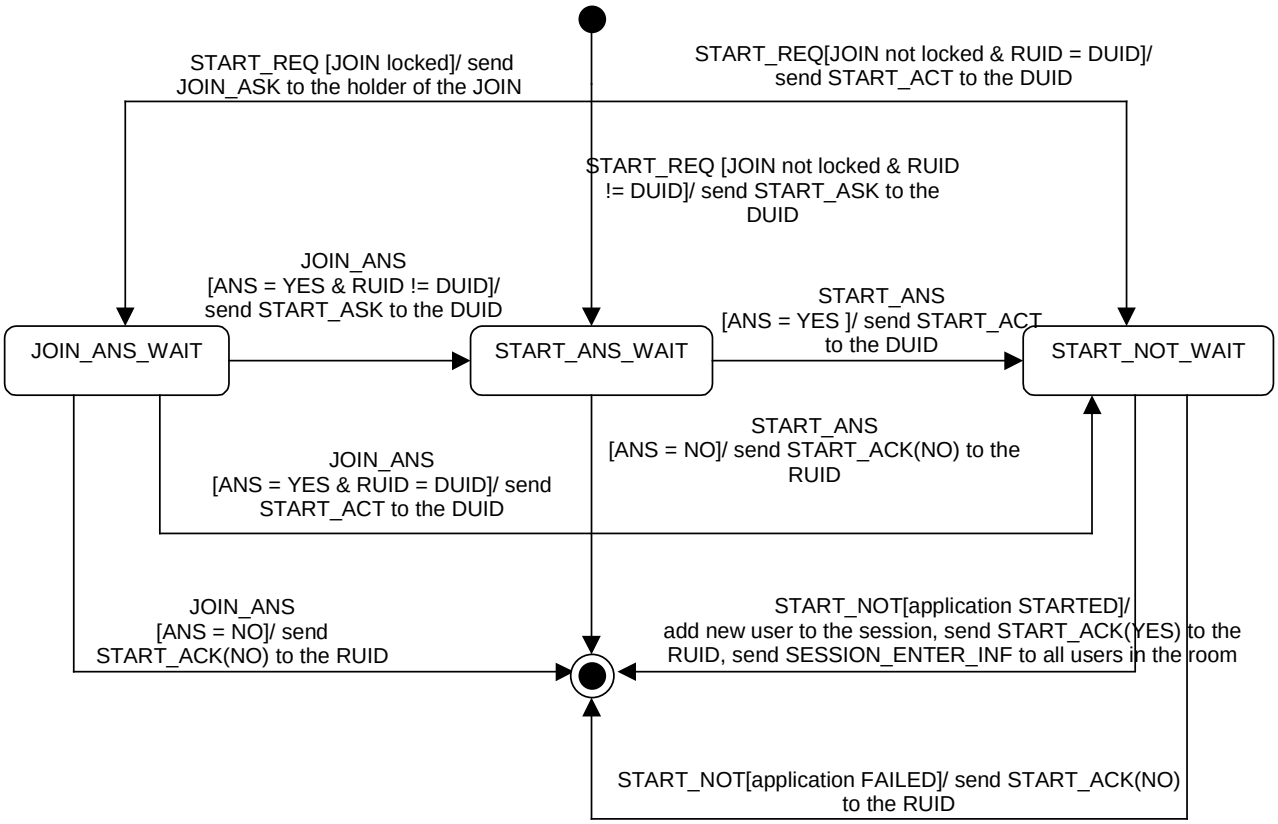

Figure 17:Dynamic model of the session join operation

It's important to point out that the change of the state of the system (adding the new user to the session) is postponed as much as possible so that the irregular termination of the operation will not put the system in the inconsistent state. The information about the new user is introduced only when the operation is successful.

6 CONCLUSIONS

Message routing and state management mechanisms differ in various collaboration models. In the synchronous collaboration systems the changes of the system state introduced by the users must be immediately visible to the others. It concerns the state of

the collaborative applications as well as the state of the collaborative framework. Such requirement imposes significant time constraints on the message distribution implementation. Moreover, the synchronous collaboration system must maintain dynamic state that describes established sessions and provide the users with the tools for the session creation and termination. Various collaboration applications executed in the synchronous environments have different needs as far as the session management and message distribution is concerned. As a result, different session models and communication channels with different characteristics are available to satisfy their requirements.

Asynchronous collaboration does not have such strict time constraints as far as the message distribution is concerned. The main requirement is the high reliability of the communication channels used by the system. The environments that maintain the shared content must provide mechanisms for the shared content modification. The data adding is preferred over data changing because it allows storing the history of the shared content modification. State of the asynchronous environment is persistent in order to provide protection against system failures that may interrupt the collaborative session. Information about changes in the system state is distributed among the users using notification or polling mechanisms.

Also the system architecture has significant impact on the message routing and the state management. The centralized systems store the system data in one place what makes the state management much easier. Complex collaboration mechanisms such as shared objects or floor control may be implemented by the systems that are based on the centralized architecture. Messages from applications are sent to the one point and then redistributed so

that ordering and reliability may be provided. However, the centralized system are not well scalable and cannot handle real-time data streams.

The server network architecture offers better scalability because it divides the load placed on the system among predefined number of servers. One server provides services to the local community and enables communication with the remote users by contacting other servers in the network. Unfortunately, such approach introduces several new problems. Systems based on the server network architecture need special mechanisms to ensure the state consistency. Also message distribution in such systems requires maintaining additional route information.

Finally, fully distributed architecture is employed mainly by the systems that handle multimedia streams. This solution provides the best scalability because all data are sent directly to the destination. However, such approach makes the state management difficult. As a result, peer-to-peer systems offer limited session and shared state management functionality. Also session establishment process poses significant problems. In order to provide users with the information about the sessions, the specific directories must be employed. Significant progress in the area of fully distributed environments may be expected when the reliable multicast protocols are introduced.

An interesting solution to the problems that affect the presented architectures seems to be an architecture that provides centralized approach to the state management and distributed approach to the application data sending. Such solution is adopted by the Tango2 system.

The Tango2 system provides support for both forms of collaboration: synchronous and asynchronous. Therefore it must take into account demands of both models. Tango2 provides collaborative framework with support for the session management, shared application state, reliable data channels and session recording. Using this framework, the customized collaborative environments may be constructed and various session models may be employed. The collaborative applications have access to the several types of communication and synchronization mechanisms. The HTTP servers provide the application content so that the collaborative server may handle control tasks and application shared state. The applications that want to send large data streams may establish multicast sessions using collaborative infrastructure provided by the system.

7 BIBLIOGRAPHY

[1] "Welcome to HyperNews", http://union.ncsa.uiuc.edu/HyperNews/get/hypernews.html.

[2] Brian Kantor, Phil Lapsley, "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News", RFC 977, U.C. San Diego and U.C. Berkeley, 1986.

[3] Jonathan B. Postel, "Simple Mail Transfer Protocol", RFC 821, Information Sciences Institute, University of Southern California, 1982.

[4] D. LaLiberte and A. Braverman, "A Protocol for Scalable Group and Public Annotations", http://union.ncsa.uiuc.edu/~liberte/www/scalable-annotations.html.

[5] Lukasz Beca, Gang Cheng, Geoffrey C. Fox, Tomasz Jurga, Konrad Olszewski, Marek Podgorny, Piotr Sokolowski and Krzysztof Walczak, "Java Enabling Collaborative Education, Health Care, and Computing, Concurrency: Practice and Experience, Vol. 9(6), 521-533, June 1997.

[6] Alex Pang, Craig Wittenbrink, "Collaborative 3D Visualization with CSpray", IEEE Computer Graphics and Applications, 32-41, March-April 1997.

[7] Daniel Garfinkel, Bruce C. Welti, Thomas W. Yip, "HP SharedX: A Tool for Real-Time Collaboration", Hewlett-Packard Journal, 23-36, April 1994.

[8] K. Holtman, "The Futplex System", Proceedings of the ERCIM workshop on CSCW and the Web, Sankt Augustin, Germany, February 7-9, 1996, http://orgwis.gmd.de/projects/W4G/proceedings/futplex.html

[9] Sandeep K. Singhal, Binh Q. Nguyen, Jimmy Nguyen, Richard Redpath, Michael Fraenkel, "InVerse: Designing an Interactive Universe Architecture for Scalability and Extensibility", Precedings of the 6th International Symposium on High Performance Distributed Computing (HPDC'97), 61-70, 1997.

[10] Hall, Robert W., Mathur, Amit G., Jahanian, Farnam, Prakash, Atul, and Rasmussen, Craig, "Corona: A Communication Service for Scalable, Reliable Group Collaboration Systems", The Proceedings of the Sixth ACM Conference on Computer-Supported Cooperative Work (CSCW 96), Boston, MA, November 1996.

[11] Jang Ho Lee, Atul Prakash, Trent Jaeger and Gwobaw Wu, Supporting Multi-User, Multi-Applet Workspaces in CBE , Proceedings of the Sixth ACM Conference on Computer-Supported Cooperative Work, 344-353, Boston, Massachusetts, November 1996.

[12] R. Bartle, "Interactive Multi-User Computer Games", ftp://ftp.ccs.neu.edu//pub/mud/docs/papers/mudreport.ps.gz, December 1990.

[13] Lakshman Y.N., "TechTALK: A Web based system for mathematical collaboration", http://www.mcs.drexel.edu/~ylakshma/paper.html.

[14]"NCSA Habanero Project",
http://www.ncsa.uiuc.edu/SDG/Software/Habanero/.

[15]J.Oikarinen, D. Reed, "Internet Relay Chat Protocol", RFC 1459, May 1993.

[16]"InSoft: OpenDVE Architectural Overview", InSoft, 1995.

[17]M. Handley, J.Crowcroft, C. Bormann, J. Ott, "The Internet Multimedia Conferencing Architecture", Internet Draft, ISI, UCL, Universitaet Bremen, July 1997.

[18]S. Deering, "Host Extensions for IP Multicasting", RFC 1112, Stanford University, August 1989.

[19]James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, "Object-Oriented Modeling and Design", Prentice-Hall, NJ, 1991.

[20]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, 1994.

8 VITA

NAME OF AUTHOR: Lukasz Michal Beca

PLACE OF BIRTH: Plock, Poland

DATE OF BIRTH: May 14, 1973

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

Technical University of Warsaw, Warszawa, Poland

Franco-Polish School of New Information and Communication Technologies, Poznan, Poland

PROFESSIONAL EXPERIENCE:

Graduate Research Assistant, Northeast Parallel Architectures Center, Syracuse University, 1996-97