Stuttgart University
Distributed Systems Dept
Breitwiesenstraße 20-22
D-70565 Stuttgart, Germany

St. Petersburg Technical University
Distributed Computing & Networks Dept
Politehnicheskaya ul. 29
195251 St. Petersburg, Russia

Configuration and Synchronization Management in Distributed Multimedia Systems

CR-Klassifikation: C.2.4, C.4, D.4.7, G.1.6, G.2.2,, I6

Technical Report of the Project OPTIMUS 01 IR 605
'Optimization, Modeling and Implementation of Distributed Multimedia Systems'

Prof. Dr. K.Rothermel
Responsible Head of the Project

Distributed Systems Dept
Stuttgart University

Prof. Dr. Yu.Karpov
Responsible Head of the Partner Research Group

Distributed Computing and Networks Dept
St. Petersburg Technical University

Fakultätsbericht 1997/11
Technical Report
August 1997

Fakultät Informatik
Institut für Parallele und
Verteilte Höchleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Abstract

This annual technical report presents the current state of the project OPTIMUS 01 IR 605 'Optimization, Modeling and Implementation of Distributed Multimedia Systems', which is a research activity at St. Petersburg Technical University, Distributed Computing and Networks Department in collaboration with Stuttgart University, Distributed Systems Department. The project is supported by a research grant of the 'Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie' of Germany[1].

The project is dedicated to the problems of configuration and synchronization management modeling, analysis, optimization and development for distributed multimedia systems.

The annual report presents the methods, models, algorithms, mechanisms, and a simulation tool developed for analysis, optimization and design of distributed multimedia application configuration management, negotiation and resource reservation protocol, and adaptive synchronization protocol, that support configurable distributed multimedia applications. Results of computational experiments conducted for complexity evaluation of algorithms proposed are presented as well. Problems of testbed platform development and implementation for configuration and synchronization management of distributed multimedia systems are discussed.

The research results obtained are oriented on the further development of CINEMA (Configurable IntEgrated Multimedia Architecture) that was designed at the Distributed Systems Department of Stuttgart University and supports development and control of multimedia applications with arbitrary processing topologies consisting of multiple data sources and sinks as well as arbitrary intermediate processing stages.

[1] Das diesem Bericht zugrundeliegende Vorhaben OPTIMUS wurde mit Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie unter dem Förderkennzeichen 01 IR 605 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Executors

Distributed Systems Department
University of Stuttgart
Stuttgart
Germany

Kurt Rothermel, Prof., Dr.
Responsible Head of the Project

Gabriel Dermler, PhD

Walter Fiederer, PhD

Distributed Computing & Networks Dept
Technical University of St. Petersburg
St. Petersburg
Russia

Yuri Karpov, Prof., Dr.
Responsible Head of
the Partner Research Group

Alexander Hagin, Prof., Dr.
Responsible Research Executor

Igor Chernorutskij, Prof, Dr.

Yuri Senichenkov, Assoc. Prof., Dr.

Alexander Glebovsky, Assoc. Prof.

Andrei Borshev, Dr.,

Viktor Roudakov,

Gennadiy Shchemelev,

Nikolaj Kolesnikov,

Vladislav Voinov, PhD

CONTENTS

1 Introduction

This annual technical report presents the current state of the project OPTIMUS 01 IR 605 ‘Optimization, Modeling and Implementation of Distributed Multimedia Systems’, which is a research activity at St. Petersburg Technical University, Distributed Computing and Networks Department in collaboration with Stuttgart University, Distributed Systems Department. The project is supported by a research grant of the ‘Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie‘ of Germany.

Distributed multimedia applications (DMA) such as multimedia mail, collaborative work systems, teleconferencing, kiosks, virtual reality applications and others require high-speed computer networks with high processing and transfer rate, adaptive, lightweight transmission protocols on top of the network.

The usage of multimedia was enabled by the technological progress in the fields of networks and endsystems that supply needful equipment for an even increasing transmission ant computational power. However, the functionality of modern operating and transport systems is not sufficient to allow the efficient development, implementation and control of distributed multimedia applications. A multimedia development platform is required to enrich the given functionality by specialized abstractions and mechanisms that support distributed multimedia processing. The main issues that have to be tackled are communication and processing of multimedia data under real-time conditions, the automatic placement of DMA in distributed computer systems (DCS), resource reservation, and synchronization of multimedia data streams.

The project is dedicated to the problems of configuration and synchronization management modeling, analysis, optimization and development for distributed multimedia systems. The goal of OPTIMUS project is as follows:

- to develop an approach, models, optimization methods and algorithms for configuration management based on automatic mapping of a distributed multimedia application (DMA) to a distributed computer system (DCS) and negotiation and resource reservation protocol,

- to develop the adaptive synchronization protocol and elaborate simulation models for its analysis and optimization,

- to develop a testbed platform to probe and evaluate algorithms and mechanisms proposed for DMA management

The annual report presents the methods, models, algorithms, mechanisms, and a simulation tool COVERS developed and used for analysis, optimization and design of DMA configuration management, negotiation and resource reservation protocol, and adaptive synchronization protocol, that support configurable distributed multimedia applications. Results of computational experiments conducted for evaluation of complexity of algorithms proposed are presented as well. Problems of testbed platform development and implementation for configuration and synchronization management of distributed multimedia systems are discussed.

The research results obtained are oriented on the further development of CINEMA (Configurable IntEgrated Multimedia Architecture) that was designed at the Distributed Systems Department of Stuttgart University and supports the development and control of multimedia applications with arbitrary processing topologies consisting of multiple data sources and sinks as well as arbitrary intermediate processing stages.

The remaining part of the report is organized in the following way. In Section 2, the DMA lifecycle and management tasks at each stage are described, the problems concerning the paper are formulated. In Section 3, architecture and models of CINEMA system for configuring and controlling DMA are presented. In Section 4, negotiation requirements, architecture and an algorithm of configuration service are described.

In Section 5, the problem of mapping a DMA to a DCS is discussed and formulated, mapping algorithms that differs by computational complexity and solution error are proposed, experimental results of algorithms' complexity analysis are presented and discussed.

In Section 6, the adaptive synchronization protocol developed at the IPVR is considered. A formal specification and model of the protocol are described using the abstractions, high-level language, and functionality provided by COVERS software platform elaborated by the research team of the Technical University of St. Petersburg. The protocol performance evaluation and optimization problems are formulated.

In Section 7, a prototype of videoconferencing system as base of a designed platform for configuration and synchronization management is described.

In Conclusions, the main results obtained are summarized. In Appendix, mapping problem specification language developed and implemented in C++ is represented.

2 Application lifecycle

To get better understanding of management tasks concerning a DMA we consider the following application lifecycle a DMA follows after it's installed on a computer system (Figure 2.1).

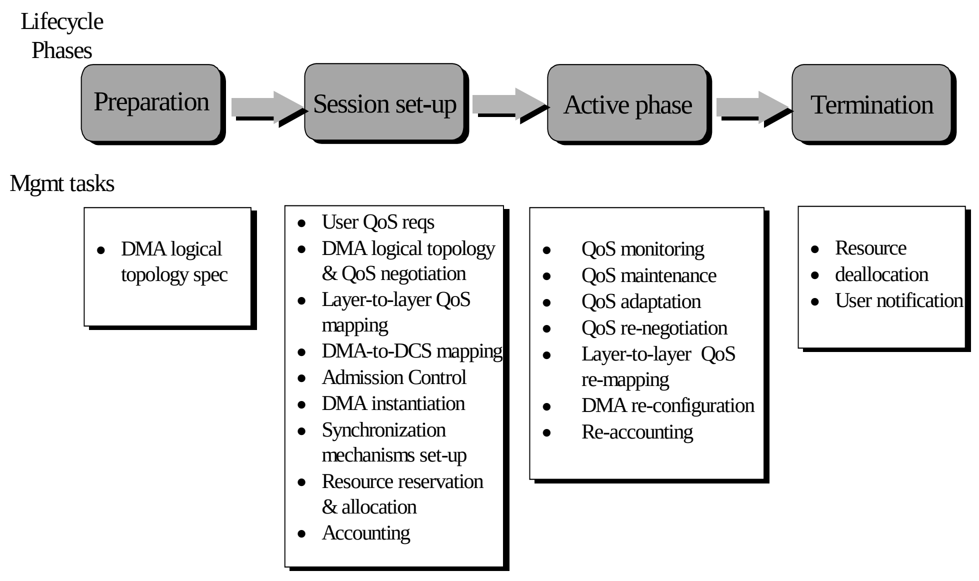


Figure 2.1 Distributed Multimedia Application Lifecycle and management tasks

We suggest to consider the following four steps during the lifecycle of a DMA session: the preparation phase, the session set-up phase, the active phase and the termination phase. To provide a requested service, the management system should analyze the correctness of a user request and specify DMA logical structure (topology) at the preparation phase, to establish a DMA session with QoS requested by the user, to control and to maintain the session QoS during the active phase, and to terminate the session when requested by the user or due to a failure, e.g. during heavy network congestion.

Let us give a brief description of DMA management tasks needed during these four phases.

2.1 Preparation phase

Initially a user specifies a DMA logical topology he wants to utilize. The user also can be asked to specify how he is going to work with the application - supposed session time (i.e. how long), start time (immediately or at any time in the future), etc. The user may declare some QoS management policies, such as what the system should do if QoS degrades below a particular level - terminate the application, try to automatically adapt QoS without user intervention or negotiate new QoS with the user, - what QoS requirements the user is ready to weaken, if all the requirements can't be satisfied all together, etc. Doing that the user and the system prepare a contract, in which the user is obliged to use the application in a certain manner, and the system has to provide a certain level of QoS in turn. The contract assignment will be done in the next phase, when DMA will be placed in the DCS and user QoS requirements will be negotiated with the DMA topology.

DMA logical topology specification is concerned with defining DMA topology as a set of components interconnected via links. Components are processing elements encapsulating functions for capturing, storing, presenting and manipulating continuos data streams. A link is an abstraction of an unidirectional communication channel, conveying stream from one component to one or more others.

The specified DMA topology is checked syntactically and semantically for media stream compatibility between interconnected components.

2.2 Establishment phase

After DMA user submitted logical topology and the topology was checked, the session set-up phase begins.

User specifies QoS and stream synchronization requirements, the sources and sinks of the DMA, that have to be established at a time. The management system negotiates the QoS requirements of the user and QoS constraints of the DMA topology, finds a DMA placement in the DCS, defines required resources needed to support desired QoS, reserves the resources, prepares all mechanisms needful for streams' synchronization. After that, the session can be initiated at the given start time by launching the DMA and its associated clock.

Additionally the user may require in advance the fixed allocation of particular DMA components (usually sources and sinks) in the DCS, i.e. which components have to be placed to which computers.

User QoS requirements are concerned with defining required levels of QoS for the DMA user(s). Required QoS values may be expressed in terms of advisory values, mandatory values, upper and lower thresholds or a variety of other forms.

Layer-to-layer QoS mapping performs the translation between representations of QoS at different system levels. In particular, this mapping function is required to translate the user QoS parameter values to those of the service provider to perform resource reservation as required to support the requested QoS.

DMA logical topology & QoS negotiation permits to refine a DMA topology, which might support the requested QoS. QoS requirements of the DMA user(s) have to be mapped to each component of the DMA. So, the problem is to decompose the QoS requirements of the DMA users to QoS requirements to each component of the DMA taking into account the DMA topology, functional capabilities and resource availability of the DCS.

Moreover, because a DMA can have many users with different conflicting QoS requirements, the management system analyses correctness of every user request and negotiates the QoS requirements of the different DMA users with each other and with the DMA topology. When user specifications are submitted, the management system is to check for inconsistencies among the requirements of a single user as well as the whole user community. If the system cannot provide requested QoS for a user, it should re-negotiate QoS requirements with the user or refuse the user request.

DMA-to-DCS mapping permits to find the optimal placement plan of the DMA topology into a given DCS taking into account the resources required by the DMA and the DCS resources available in an offered time interval of the DMA session.

Admission control is responsible for comparing the resource requirements that arise from the QoS levels associated with a new DMA, and the available resources in the DCS.

Synchronization mechanisms set-up allows controlling and synchronizing the data flow of data streams arriving at sink components.

DMA instantiation is concerned with loading DMA component code.

Resource reservation & allocation is usually necessary in order to support a given level of QoS during DMA session. While resource reservation is done during *DMA-to-DCS mapping* operation, allocation of the resources is conducted after DMA instantiation, i.e. just before the beginning of the active phase.

Accounting concerns with the evaluation of the costs relative to a service requested by a user.

2.3 Active phase

The active phase starts at the time specified by a user, i.e. immediately or at any time in future. In the last case, the management system provides advance reservations for guaranteed and predictive service as shown in [DKPS95], [FGV95], [WDS$^+$95].

At the active stage the application is launched providing certain services to its users. The management system should support QoS monitoring and maintenance, if QoS requirements are violated.

QoS maintenance is based on dynamic mechanisms (e.g., fine grained resource tuning strategies), which allow to ensure that the required performance of individual DMA components as well as of DMA as the whole are kept within given bounds.

QoS monitoring is used to allow each layer of the system to track the ongoing QoS levels and compare them with the initial QoS requirements.

QoS adaptation implements coarse grained QoS maintenance control and must be able to exhibit graceful degradation reacting adaptively on changes in the environment and to violations of the contracted QoS.

QoS re-negotiation, layer-to-layer QoS re-mapping, DMA reconfiguration and re-accounting are performed if the DMA performance violates the contracted QoS.

2.4 Termination phase

It’s time when all the resources submitted to the application (or its part) should be freed and the users should be notified about session termination. The latter is important when application termination was initiated not by the users, but by the system due to the certain reasons (allowed service usage time has elapsed, or an unrecoverable error occurred). *Resource deallocation* procedure sends the notifications to all entities involved in the QoS provision relative to this DMA session to free reserved resources.

2.5 Concern of this paper

Undoubtedly, being different all the DMA management tasks mentioned above have a lot of common properties. To perform any of them one should have dependencies between QoS parameters, which comprise a QoS goal, and system parameters at various levels of system architecture. Expressed in some form these dependencies are interpreted specifically for a particular management task (e.g. monitoring, diagnosis or operational control) and management policies. For instance, in papers [AH+97], [AH+97a], the active phase of the distributed application lifecycle mostly is considered, when all application components are attached to the resources, and management architecture, model based approach for QoS monitoring and operational control schemes for the World Wide Web system are proposed.

In our paper, we will discuss mostly the preparation and session set-up phases, algorithms and protocols used for management tasks of these two DMA lifecycle phases.

3 CINEMA system for configuring and controlling distributed multimedia applications

3.1 Application model

In this section we briefly introduce abstractions for modeling configurable distributed multimedia applications. Similar concepts are pursued by various research groups [KHSM95], [BCA$^+$92] including the group defining IMA MMS [IMA93]. Here we describe the terminology used for CINEMA (Configurable INtEgrated Multimedia Architecture) development platform. For detailed description of CINEMA we refer to [RBH94], [RDF97].

DMA are employed to generate, process and consume (e.g. present) continuous (e.g. audio, video) data streams over distributed locations into computer network. Generally, a DMA consists of some or more components, interacting with each other by data streams [RBH94]. Source components generate data, filters and mixers manipulate data, and sink components present data to users. DMA can be represented by one or some data flow graphs, composed out of such components and links representing the data streams between components.

Components encapsulate processing of multimedia data, e.g. for generating, presenting or manipulating data. To provide a uniform data access point for components, ports are used that deliver data units to the component (input port) or take data units from the component (output port). Source components are associated with output ports only, sink components – with input ports only. Intermediate components receive data from a number of input ports, perform some operation on the received data, and send the result via a number of output ports.

An application is constructed by specifying a topology of components interconnected via links. A link indicates a data flow between two interconnected ports and provides an abstraction from underlying communication mechanisms that may be used to perform the transport of data units.

3.2 CINEMA system architecture

CINEMA [RBH94] is a middleware system that provides high abstractions and services for configuration distributed multimedia applications, the set-up of communication sessions along with QoS negotiation and resource reservation, as well as the service for data flow control and synchronization. CINEMA assists the client in these services in an integrated way.

Architecture of CINEMA system is depicted in Figure 3.1.

A client is a program entity that realizes, on one hand, a graphical interface service towards human end-users and, on the other hand, performs DMA topology construction, session set-up and data flow control using the services offered by CINEMA.

In CINEMA, a configurable distributed multimedia application is defined as a client containing the description of a topology of interconnected components, temporal properties of the topology and a session. The session is defined to select parts of the topology, that shall be established at a time, and to specify a media-specific and end-to-end QoS parameters corresponding to a QoS requested by end-user. The QoS description is located at the input ports of sinks.

In [RBF96], [RDF97] and [Bar96], configuration and session management services were defined assuming that a location of a DMA in a DCS is specified in any way in advance (e.g., by end-user). Now we consider extended configuration management that includes the automated mapping a logical application topology to a computer network. Adding the mapping function causes some rearrangements of the previous function distribution between the configuration and session services.

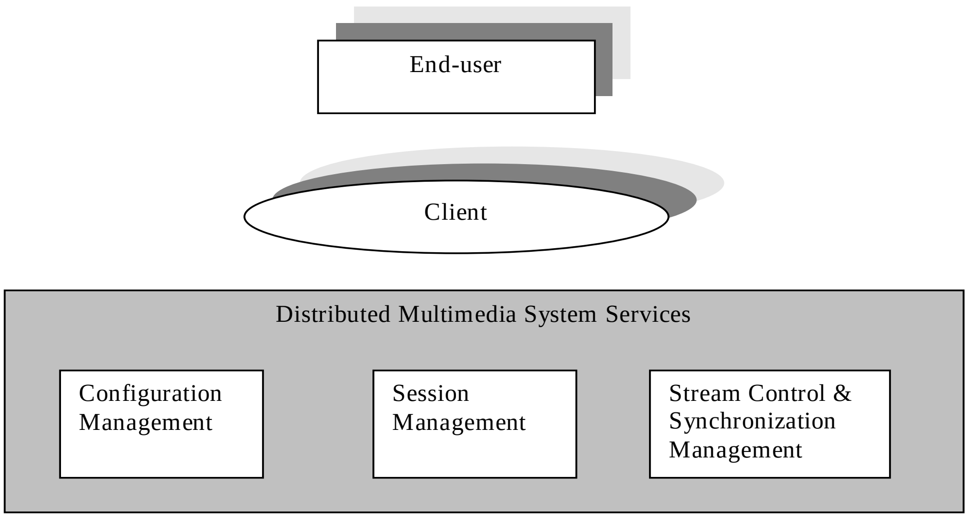

End-user
Client
Distributed Multimedia System Services
Configuration Management
Session Management
Stream Control & Synchronization Management

Figure 3.1 Architecture of CINEMA system

The configuration process is executed during the preparation and session set-up phases. First, the client refers to existing components by globally unique names and interconnects components to a logical application topology. Each stream and all ports connected to the stream are associated with one media type, e.g. “MPEG_encoded_video”. A specified DMA logical topology is checked syntactically and semantically (for compatibility between interconnected ports) by the configuration service.

The second step of the configuration process is initialized when the client starts a session set-up. During the session set-up phase, the components necessary for the session (may be not all ones of the DMA) are selected. Desired QoS has to be indicated to CINEMA. Stream media values between the component ports are negotiated corresponding to the requested QoS taking into account a set of acceptable computers, their functional capabilities and resource availability. Then an optimal placement of the DMA in the DCS is found, resource reservation for the components and links is executed, and the physical application topology is instantiated. The instantiation concerns the provision of runnable code for components and links and the establishment of communication channels required for managing subsequent services.

Synchronization management is used within a session to control the data flow of data streams arriving at sink ports. CINEMA offers VCR-like commands to start, stop, and scale data flow of such streams. In addition, CINEMA allows to group streams for atomic control and to specify synchronization requirements between several streams by offering the concept of clock hierarchies [RH96a].

A stream synchronization service frees a client from the ever-recurring task of implementing flow control and synchronization algorithms as part of the development process of each multimedia application. By using the middleware service for stream synchronization, time-dependent data streams can easily be integrated into various applications, such as CSCW applications or tools to present multimedia documents.

3.3 QoS architecture

The necessity of different QoS levels arises from different tasks that have to be solved by a user and by a service provider. At highest level, end-user oriented QoS abstractions must be allowed to express user QoS requirements. At the lowest level, QoS descriptions serve to indicate resource demands. Consequently, the lowest level QoS description is resource oriented and independent on media-specific characteristics.

Besides at the middle level, media-specific QoS parameters are needed to negotiate QoS. The goal of the QoS negotiation is to settle on the best QoS complying with client (user) QoS request under two kinds of restrictions: limited functional capabilities of computers that are used for allocation and processing DMA components, and limited resource availability for links and components.

In CINEMA, three levels of QoS are considered: client level QoS (C-QoS), session level QoS (S-QoS) and transport level QoS (T-QoS).

At the highest level, C-QoS is specified in a manner appropriate to end users, and hence is end-user-specific.

The QoS specified at the session level is media-specific. The media parameters of the stream’s type and a number of generic parameters, such as delay, jitter and loss rate, define S-QoS of a stream. The client is responsible to map the C-QoS specified by the end user to the S-QoS at the session level. Of course, for a given type of S-QoS, various types of C-QoS and mappings are conceivable.

At the transport level, T-QoS is specified in a media independent fashion, e.g., such parameters as packet size, packet rate, burst size, delay, and jitter. In CINEMA, the mapping from media-specific QoS parameters to the T-QoS parameters is done within the link objects.

During session establishment, flowspecs are used to convey QoS information at the session and transport level. At the transport level, a flowspec carries media-independent parameters and is communicated along the selected network path. At the session level, a flowspec contains media specific and generic parameters and is passed along the flow graph structure, where both links and components are involved in QoS negotiation.

4 Configuration service

4.1 Negotiation requirements

Before discussing the configuration management, we consider DMA topology peculiarities, which cause QoS dependencies across the application topology. These dependencies have to be taken into account to construct QoS negotiation and mapping algorithms and protocols of the configuration service.

4.1.1 Functional capabilities and resource shortages

Functional capabilities of a DMA component to support different QoS levels are restricted. Limited functional capabilities of components depend on their design. This kind of capabilities has to be taken into account just at the preparation phase of the DMA life cycle.

Moreover, different computers can implement components in various soft- and hardware ways, thereby it may also influence the component functional capabilities that are practicable. So, functional capabilities of DMA components may depend on functional capabilities of computers (their soft- and hardware) on which they are placed.

Resource shortage somewhere in a DCS (e.g. on the computer) forces a component or link (located in the computer) to support QoS below their functional limitations.

4.1.2 Port type compatibility

In distributed multimedia applications, data streams are typed. Each media type defines a set of so-called media parameters, which specify the characteristics of a particular stream instance. As streams are communicated via ports, ports objects are typed and can only be linked if they are of compatible type. Such port type checking is performed during the preparation phase when application logical topology is instantiated.

4.1.3 Format constraints of media parameters

As ports are typed, only a certain type of stream can be communicated via a given port. However, the instances of a particular stream type may significantly differ in their media parameter values. It cannot be assumed that a component port supports the entire value range for each media parameter.

To express the capabilities of components with regard to the stream formats they are able to consume and/or produce, so-called ***format constraints*** are introduced in [RDF97]. Each component port is associated with a media that is characterized by appropriate media parameters. Each media parameter has a (possibly empty) set of format constraints, which indicate the value range of such parameter that can be supported by the component at that port.

Format constraints for each media parameter associated with each component port depend on functional capabilities of computer, on which the component is placed. Hence, unlike XNRP, format constraints relate both to components and computers, and they have to be specified for each (component, computer) pair.

4.1.4 Stream relations

In addition to format constraints, so-called ***stream relations*** between ports of a same component have to be considered by the negotiation process [RDF97]. For example, an audio mixer mixing several audio streams usually requires the same quality to be delivered at all input ports.

For fixed filter, a stream relation between input and output ports is as follows:

*param: @outport = F * @inport.*

Such stream relation means that the value of media parameter at input port is to be F times greater than the value of this parameter at the output port. Here F denotes the filter factor.

For mixer components, mixing stream relations are defined between pairs of input ports *(i,j)*:

param: @inport_i = $f_{i,j}$ ** @inport_j.*

The relation means that the value of media parameter at input port *i* is to be $f_{i,j}$ times the value of this parameter at input port *j*. For the mixer with *n* input ports, it is enough to specify *n-1* factors $f_{i,j}$ relative to any input port *j*.

Mixing relation between the output port and an input port of a mixer

param: @outport_k = $f_{k,j}$ ** @inport_j.*

Usually, input port *j* of the mixer is chosen such that $f_{k,j} = 1$ and such input port *j* is used for specification other *n-1* input ports. Hence, to define all stream relations between input and output ports of a mixer, it is enough to specify number of input ports and number j of the input port, relative to which filter factors of other input ports are specified. Such input port *j* of a mixer is called *key port.*

Obviously, the stream relations defined at a component not only affect the media parameters of this component's input and output streams but impact the entire flow graph.

4.1.5 Topology complexity

Introducing mixing components and multicasting links into application topology makes the topology more complex. Such components and links complicate the negotiation process and cause additional QoS dependencies that reduce QoS ranges for which the negotiation process will be successful. To isolate and to reduce the impact of such QoS dependencies, the concept of special components, termed variable filters, is proposed in [DFR95], [DFBR95].

In effect, a variable filter decouples QoS provided at its input port from QoS delivered at its output port. This feature of the variable filter allows negotiating QoS for the DMA topology closer to QoS requirements and raises DMA session flexibility for "heterogeneous" QoS requests reflecting the varying QoS requests of independent (human) end-users supported by the client.

4.1.6 Requirements to the configuration service

Above arguments laid the ground for requirements to the configuration service for distributed multimedia applications:

- Support for media specific QoS parameters,
- Support for port type compatibility for DMA logical topology (at the preparation phase),
- Support for various QoS due to limited functional capabilities of components, computers and resource availability of the computers,
- Support for media parameter format constraints associated with each component port. Unlike NRP [DFBR95] and XNRP [RDF97], the format constraints relate both to components and computers, and they have to be specified for each (component, computer) pair,
- Support for stream relations,
- Support for complex DMA topologies including mixing components, variable filters and multicasting links.

All these requirements have to be taken into account for development of the configuration service (realized by Configuration server).

4.2 Configuration service architecture

The configuration service is divided into three parts. One part, called application handler [Bar96], is located on the system the client is established and services the client to build up DMA logical topology as a flow graph at the preparation phase (see Figure 2.1). Application handler checks syntactically and semantically the topology, e.g., port type compatibility.

The second part, called configuration server, can be organized as a centralized entity. It starts at the session set-up phase (see Figure 2.1) and defines DMA physical topology including:

- selecting all DMA components that are part of session creation,
- defining format constraints for each component port,
- pre-negotiating media values between component ports for given QoS requirements taking into account format constraints and a set of acceptable computers for each component,
- mapping pre-negotiated DMA topology to the DCS,
- resource reservation for components, and sending and receiving sides of links,
- loading and instantiating components.

The third part, we call configuration handler following [Bar96], is based on XNRP [RDF97] and is distributed over the computers that have been used for allocation of DMA components. The configuration handler completes the session set-up phase including following operations:

- final QoS negotiation taking into account communication resource demands of the links and resource availability of the transport network,
- final computational and communication resource reservation,
- distribution of components on threads,
- preparation of all mechanisms necessary for stream synchronization.

At the termination phase (see Figure 2.1), configuration service selects all components that are part of session deletion, terminates the session and releases all resources occupied by the components.

The algorithms for selecting components, which are part of session creation and deletion, as well as for distribution of components on threads are proposed in [Bar96].

Let us discuss the configuration server in detail.

The main complexity to develop an algorithm of the configuration server is the interdependency of the QoS negotiation and DMA mapping processes. Actually, as argued above, format constraints relate both to components and computers, and they have to be specified for each (component, computer) pair. So, to negotiate QoS, format constraints for each port of every component must be given. To define format constraints, the functional capabilities and resource availability of computers, on which the components will be placed, have to be given, i.e. the components must be pre-attached to computers. However to find an acceptable or an optimal placement of components on computers, required resources of components and links are necessary. To define required resources, negotiated values of media parameters at each port corresponding to QoS requirements have to be given. Thus the “closed circle” is got that verifies the interdependency mentioned above.

To construct a configuration server algorithm, we base on approaches, methods and algorithms for QoS negotiation implemented in NRP [DFBR95] and XNRP [RDF97], and for DMA to DCS mapping developed in [HRD96], [HRD96a], [IH96].

It is important to note that QoS negotiation algorithms deal with QoS value ranges, i.e. format constraints. And they compute one value of each media parameter at every port in the DMA topology. Such set of point values obtained, on one hand, minimizes resource consumption, and on the other hand, supports the highest possible QoS values in the acceptable intervals of QoS requirements specified by the client at sink components.

In opposite side, at the beginning, the mapping algorithms deal with one value of resource demands for each resource type for every DMA component and link, and also with point values of available corresponding resources of a computer network. And they try to find an optimal or quasi-optimal placement of the DMA in the computer network.

Hence, the problem is to define initial format constraints as input data for the QoS negotiation algorithm. Then the point media values obtained by the algorithm are transformed to resource demands, which are used as input data by the mapping algorithm for DMA placement in the DCS.

The proposed approach for definition of initial format constraints is as follows. Configuration server asks every computer what kind of components and what format constraints for each component type it can support. Then, for each component of the DMA, a set of acceptable computers is created. After, **the aggregated format constraints** are defined for ports of each DMA component taking into account the corresponding set of acceptable computers and their format constraints.

When QoS negotiation algorithm will obtain the negotiated media values, these values will be used for refining the set of acceptable computers for every component. Obviously, every such refined set will contain not more computers than corresponding initial set.

4.3 Configuration service algorithm

4.3.1 General algorithm

General representation of the configuration service algorithm is as follows:

1. Specifying (by the client) and checking (by the application handler) a DMA flow graph.
2. Specifying (by the client) clocks for stream synchronization, all source and sink components that have to be active in a session, QoS requirements, the component placement constraints.
3. Selecting all components that will be part of the session.
4. Specifying a set of acceptable computers and aggregated format constraints for each component.
5. Negotiating media values between component ports for highest possible user QoS requirements such that each media parameter at each port is characterized by one value and aggregated format constraints are satisfied. If the QoS negotiation failed then notify the client that user request can be not satisfied, stop.
6. Refining the set of acceptable computers for each component on the base of media values obtained.
7. Deriving resource requirements for each acceptable (component, computer) pair as well resource requirements of component's ingoing and outgoing links.
8. Assigning the DMA to the DCS.
9. If the assigning is not successful then checking whether the DMA can be placed in the DCS for lowest possible user QoS. For such checking, steps 5 - 8 are repeated but for lowest user QoS requirements.

   If re-assigning failed then notify client that user-request can not be satisfied and stop, otherwise any policy can be used to improve the DMA assignment obtained.
10. Reservation of computer resources needed for components as well as for its sending and receiving links, loading and instantiating the components.
11. The configuration handler based on XNRP executes the operations mentioned in 4.2.

The configuration server executes steps 3 – 10.

4.3.2 Input data for the configuration server

Configuration server collects following information
1. From client:

- flow graph, including
  a) both all components (their classes and their port types) and links (adjacent components as well as type of their output and input ports connected by the link) specification,
  b) for each fixed filter, filter factor,
  c) for each mixer
    - number of input ports,
    - key input port,
    - filter factors of input ports relative to the key port,
- QoS requests specified for input ports of sinks,
- the set of (pre-attached component, computer) pairs defining what components and to which computer they are assigned in advance (these are usually sink and source components).

2. From computers:
   - set of component types supported by each computer (as result of admission control using information of functional capabilities and resource availability of the computers),
   - format constraints for each (component type, acceptable computer) pair,
   - resource requirements for (component, its negotiated media values, acceptable computer) triple,
   - resource requirements for sending side (output port of one adjacent component) and receiving side (input port of other adjacent component) of each link,
   - resource availability,
   - cost of each resource type.

The configuration server asks and uses the data[1] from the client at step 3 (see 4.3.1).

The information about sets of supported components and format constraints is asked from computers and used at step 4 to derive aggregated format constraints. The latter is used then for QoS negotiation at step 5.

The data of resource requirements are asked from computers and used at step 7. It is important to note that QoS negotiation at step 5 makes possible to ask resource demand for each (component, its negotiated media values, acceptable computer) triple not for all values of media parameter range but only for one media value obtained at the step 5.

At step 8, the server asks and uses the data of resource availability and resource costs.

Thus, the configuration server requests the input data not simultaneously but consistently (step by step) and exactly at the step when the data is needed. Such approach allows to get actual data and take into account the results of the previous steps for revision the volume and content of the required data.

4.3.3 Refinement of the configuration server algorithm

Let us refine some steps of the general algorithm (see 4.3.1)

At step 4, the configuration server receives from each computer the set of supportable component types. Let $SupCmpo_n$ denotes the set of component types supported by the computer n. Then to specify the set of acceptable computers $AcCmpu_j$ for a DMA component j, the server includes in the set $AcCmpu_j$ every such computer n which set $SupCmpo_n$ contains the component j.

At step 4, the server receives also format constraints for each (component type, acceptable computer) pair. Assume that from all acceptable computers, the server got format constraints for a media parameter V for component j $FormCstr_j = \{(minV_n \dots maxV_n), n \in AcCmpu_j)\}$, where $(minV_n \dots maxV_n)$ denotes value range for the parameter V provided by the computer n. Then the aggregated format constraints for the component j is obtained as (minV, maxV), where $minV = min\{minV_n, n \in AcCmpu_j)$ and $maxV = max\{maxV_n, n \in AcCmpu_j)$. So, the aggregated format constraints for any

[1] To include into consideration also end-to-end QoS requirements, (throughput, delay, loss-rate), additional input data are needed, namely corresponding QoS parameter values for each (component, computer) and (link, virtual channel) pairs. In the current report, to simplify a general conception of configuration server represented, we omit the end-to-end parameters.

media parameter at any port of a DMA component are obtained by merging all format constraints from all computers acceptable for the component.

At step 5, the server negotiates media values between ports of all DMA components in two phases. In the first phase, the aggregated format constraints are “propagated” in a down stream direction from source components to sink components. The results of the phase are possible QoS ranges provided by the sources to the sinks. In the second phase, best acceptable QoS are defined for sinks taking into account the QoS ranges obtained for sinks at the first phase and QoS requirements specified by the client at the input ports of sinks. Then obtained best acceptable QoS values are “propagated” from sinks to sources to define needed exact media values at every port of components including source components. During the propagation of QoS in both phases, the aggregated format constraints and stream relations are taken into account.

At step 5, the server realizes the first and second phases of XNRP [RDF97] but without resource constraint consideration, i.e. assuming that resources are unlimited. However it is important to note, that actually resource constraints were introduced before in aggregated format constraints, and thus, they are taken into account implicitly at the step 5. For a detailed description of the two phases algorithm we refer to [RDF97].

Negotiated media values for every port of DMA components obtained at step 5 are used for refinement of acceptable computers set for every component at step 6. It means the computers of set $AcCmpu_j$ that can not provide a negotiated media value for component j must be excluded from the set $AcCmpu_j$ of computers acceptable for this component.

Mapping the DMA to the DCS executed at step 8 will be discussed in detail in the next section.

At step 9, if the mapping for the best possible QoS is not successful then the server re-negotiates the DMA logical topology for lowest possible QoS and tries to map such the DMA to the DCS.

At step 10, the obtained DMA placement in the DCS with best possible QoS or with lowest possible QoS is used by the server to complete the session set-up phase.

It is important to note the step 9 could be supplemented such that if a DMA placement for lowest QoS is found, then one can try to re-place the DMA in a way to improve QoS. Different policy for DMA re-mapping can be used. Let us consider such one.

There are given

- for lowest user QoS requirements, resource requirements of each DMA component, sending and receiving links
- for highest QoS requirements, also resource requirements of each DMA component, sending and reciving links

Hence, each component i is characterized by resource difference dR_i needed to improve user QoS from the lower level to highest one. Using the flow graph of the DMA, one can determine which sinks are influenced by component i. Let $dQoS_i$ is the sum of differences in QoS between the highest and the lowest levels for all sinks connected directly or indirectly with component i.

Then the component n with maximum value dQoSn/dRn is a start point for improving the QoS. The next acceptable higher media value of format constraints is chosen for this component, and one continues the algorithm from step 5 to renegotiate the port format constraints of DMA components and so on.

Thus we not have to do immediately with sink QoS requirements but with a component which minimum resource demand increase allows getting maximum QoS improvement.

If no acceptable DMA allocation exists for such DMA graph with new port format constraints then other component with maximum value dQoSn/dRn will be considered and so on until an acceptable DMA location for best possible QoS is found or a permissible time for the DMA mapping is finished.

5 Mapping a DMA to a DCS

5.1 Introduction to mapping problem

The general mapping problem[1] deals with the question of assigning components of a DMA to computers of a distributed system (network) so as to minimize a criterion (e.g. cost of usage of DCS resources) satisfying QoS requirements of DMA users and taking into account DCS functionality and resource constraints. The particular mapping problem deals with a DMA which components and links are specified by required computation and communicational resources that can guarantee requested QoS. Thus, in the particular mapping problem, QoS requirements are taken into account in an indirect way.

Several formulations of the mapping problem are known to be NP-Complete (e.g., [HDR96], [HDR96a]). Approximate techniques with polynomial complexity for mapping DMA, whose topology are restricted to a chain-like and tree-like structures are presented in [IH96]. In [HDR96] the mapping algorithm based on branch-bound method is proposed, and in [HDR96a] the technique for DMA initial placement improvement is considered.

There are no simple procedures, which can guarantee an optimal solution to the mapping problem due to its complexity. Therefore, heuristics techniques are employed as well to find pseudo-optimum or local optimum solutions.

Mapping techniques are usually either constructive or iterative. A constructive approach assigns the components one a time to the different computers until all components are eventually assigned. Constructive methods can be distinguished by the strategies to select the next unassigned component, and to assign the selected component.

Iterative methods transform a given complete placement into a new, hopefully improved, complete placement. If an acceptable criterion is satisfied, the new placement replaces the current one. This process is repeated until a stopping criterion is satisfied. During each iteration, a new placement is found by selecting some components and moving them to alternate computers. The initial placement needed to start the iterative process can be generated in any way, e.g. manually by user, randomly or, at last, using a constructive method.

In the paper, both constructive and iterative methods will be discussed.

Let us consider some peculiarities of the DMA topology and requirements of the real time resource management that must be taken into account for the mapping problem formulation.

The first one is the multicasting communication mode, that allows to use the capacity of a single real channel to transfer the same data to many sinks and is widely used in distributed multimedia applications. The DMA topology consisting of a single source, multicasting link and many sinks is the simplest example of multicasting application. . Multicasting logical topology can be mapped to multicasting physical topology if and only if the DCS (or some needful channels in a DCS) supports multicasting mode.

The multicasting case influences the computations of cost functions and resource usage in mapping algorithms.

Another feature of a DMA concerns, that computer resources needed for a component depend on whether adjacent components are placed in the same computer or in other one. If two adjacent components are allocated on the same computer no additional expenses are needed for their communication. In this case the capacity of required resources is determined only by processing modules of both components as soon as its link does not require the mapping to a DCS channel.

[1] The problems closed to the mapping are also addressed in literature as assignment problem (e.g. [Bok97]) or placement problem (e.g. [RPP94]). All these terms we use in our paper as synonyms.

However, should these components be placed on the different computers each of them will require, usually, transport and compression or decompression processing modules to interact. Hence, in the last case, such modules require additional computer resources that increase the total cost of DMA allocation. Thus, computation resources required by a component depend on whether the connected components are allocated to the same computer or not.

The next important property that must be taken into consideration is dynamic nature of the processes of allocatingand releasing resources performed by distributed applications, which can start and terminate atrandom moments. So, the current resource availability of DCS units and channels varies in time. It makes the practical solution of the mapping problem even more complex. There is some kind of a contradiction: on one hand, the most mapping techniques assume that the DCS resource availability is fixed until the DMA assignment solution will be obtained. On other hand, the DCS resource availability is a function of a time and depends on other applications' activities.

In [RDH96a], static, pseudo-static and dynamic mapping policies that differ in duration of blocking the DCS available resources are considered. The dynamic policy does not require blocking but allows only step by step DMA component assignment and requires any re-assignment mechanism in the case if the resource manager can not support the requested resources for the allocation of the component assigned.

The duration of blocking in static and pseudo-static policies has a threshold, its violation may cause a deterioration of behavior of applications running at the same time in the DCS. Therefore, a mapping algorithm consuming a lot of time should be manageable to be interrupted at any time, to place already assigned components in a DCS, and to resume the work from an interrupted point when the control is given back from the system. Thus, it is desirable that a mapping algorithm assigning and allocating once a component to a computer does not revise that assignment. Just a few of mapping approaches have such a property, e.g., branch bound method based approach [HDR96] does not have this property.

5.2 Mathematical formulation

The mapping problem formulation is as follows. We are given:

1. A flow graph of DMA topology including
   η – a set of components,
   $d_i(n)$ - a conditional required computational capacity for every component i (i.e. CPU cycles per second needed by the component to run normally) provided that computer n is used to assign component i,
   $m_i(n)$ - a conditional required memory amount for every component i provided that computer n is used to assign component i,
   λ - a set of links connecting components with each other, $\lambda = \{(i,j), i,j \in \eta)\}$,
   b_{ij} - a required bandwidth for every link $(i,j) \in \lambda$,
   $d_{(ij)(nm)1}$, $d_{(ij)(nm)2}$ - a conditional required computational capacity for component i (a sender) and component j (a receiver) of a link $(i,j) \in \lambda$ correspondingly to transfer (i.e. to send and to receive) data through the link provided that the components i and j are mapped to computers n and m. Parameters $d_{(ij)(nm)1}$, $d_{(ij)(nm)2}$ compose a matrix of $(|2\lambda|*|\pi|)$ elements,
   $m_{(ij)(nm)1}$, $m_{(ij)(nm)2}$ - a conditional required memory amount for component i (a sender) and component j (a receiver) of link $(i,j) \in \lambda$ correspondingly provided that computers n and m are used[1]. Parameters $m_{(ij)(nm)1}$, $m_{(ij)(nm)2}$ compose a matrix of $(|2\lambda|*|\pi|)$ elements,
   ζ_i - a set of acceptable locations (computers) for every component $i \in \eta$ in the DCS.

2. A DCS topology including
   ζ - a set of computers,

[1] In so-called homogeneous situation, when computation~~al~~ and memory resources required by every component do not depend on which computer is used for placement, the parameter n can be omitted.

- a set of virtual channels (simply channels) connecting computers with each other, $\zeta = \{(n,m), n,m \in \zeta)\}$,

R_n, M_n - an available (vacant) computation and memory resources of every computer $n \in \zeta$,

- a set of shared or/and dedicated communicational resources in the DCS[1],

ρ_{nm} - a set of communicational resources of the DCS used by channel (n,m), $\rho_{nm} \in \rho$, $\rho = \cup \rho_{nm}$,

π_s - a set of channels routed over communicational resources $s \in \rho$, $\pi = \cup \pi_s$,

A_s - a bandwidth of a communicational resource s available to the DMA, $s \in \rho$.

3. Cost functions

f - a matrix ($|\eta|*|\zeta|$) of component's allocations costs, an element f_{in} of f specifies the cost of mapping component i to computer n. If component i can not be assigned to computer n then $f_{in} = \infty$,

g - a matrix ($|\lambda|*|\pi|$) of link's allocations costs, an element $g_{(ij)(nm)}$ of g specifies the cost of mapping link (i,j) to virtual channel (n,m). If link (i,j) can not be assigned to channel (n,m) then $g_{(ij)(nm)} = \infty$,

u - a matrix ($|2\lambda|*|\pi|$) of costs for link sending and receiving sides allocations, an elements $u_{(ij)(nm)1}$ and $u_{(ij)(nm)2}$ of u specifies the cost of mapping the link sending side (output port of component i) to computer n and the link receiving side (input port of component j) to computer m correspondingly. If link (i,j) can not be assigned to channel (n,m) then $u_{(ij)(nm)1} = \infty$ and $u_{(ij)(nm)2} = \infty$.

The solution variables are x_{in} such that $x_{in} = 1$, if component I is assigned to computer n, and $x_{in} = 0$ otherwise.

The general mapping problem is to minimize the total cost of DCS resources used for DMA allocation

$$F(x_{in}) = \min \{\Sigma_{n\in\zeta}\Sigma_{i\in\eta}\, x_{in} * f_{in} + \\ + \Sigma_{(n,m)\in\pi}\Sigma_{(i,j)\in\lambda}\, x_{in} * x_{jm} * (g_{(ij)(nm)} + u_{(ij)(nm)1} + u_{(ij)(nm)2})\} \qquad (5.1)$$

subject to

every component i of the DMA have to be placed on a DCS computer and only on one computer

$$\Sigma_{n\in\zeta}\, x_{in} = 1, \forall\ i \in \eta, \qquad (5.2)$$

computational resource of each computer $n \in \zeta$ used by all components assigned to the computer, and also by corresponding sending and receiving sides of links, must not exceed the available resource of the computer

$$\Sigma_{i\in\eta}\, [x_{in} * d_i(n) + \Sigma_{j\in\eta}\Sigma_{m\in\zeta}\, x_{jm} * (d_{(i,j)(nm)1} + d_{(ji)(mn)2})] \le R_n, \\ \forall\ n \in \zeta,, i \neq j, n \neq m \qquad (5.3)$$

memory units of each computer $n \in \zeta$ used by all components assigned to the computer, and also by corresponding sending and receiving sides of links, must not exceed the available resource of the computer

$$\Sigma_{i\in\eta}\, [x_{in} * m_i(n) + \Sigma_{j\in\eta}\Sigma_{m\in\zeta}\, x_{jm} * (m_{(i,j)(nm)1} + m_{(ji)(mn)2})] \le M_n, \\ \forall\ n \in \zeta,, i \neq j, n \neq m \qquad (5.4)$$

total bandwidth of all DMA links mapped on communication resource s must not exceed the available bandwidth of the resource

$$\Sigma_{(n,m)\in\pi}\Sigma_{(i,j)\in\lambda}\, x_{in} * x_{jm} * b_{ij} \le A_s, \forall\ s \in \rho \qquad (5.5)$$

Here $g_{(ij)(nn)} = 0$ if components i and j are assigned to a same computer n, $(i,j) \in \lambda$; $g_{(ij)(nm)} > 0$ if components i and j are assigned to different computers, i.e. $n \neq m$, $(i,j) \in \lambda$ and $(n,m) \in \pi$. Similarly, $u_{(ij)(nn)1} = 0$ and $u_{(ij)(nn)2} = 0$ if components i and j are assigned to a same computer n, $(i,j) \in \lambda$; $u_{(ij)(nm)1} >$

[1] Depending on used mapping algorithm, different communication resources can be taken into account. For example, to simplify the mapping algorithm only computer interfaces and its bandwidths (and no communication links or networks) can be considered.

0 and $u_{(ij)(nn)2} > 0$ if components i and j are assigned to different computers, i.e. $m \neq n$, $(i,j) \in \lambda$ and $(n,m) \in \pi$.

It is important to note that if it is needed take into account no resources of the communication network of the DCS then one omits coefficients $g_{(ij)(nm)}$ in the objective function F and considers communication constraints (5.5) only for computer communication interfaces.

This mapping problem formulation is a nonlinear integer programming NP problem with Boolean variables.

5.3 Mapping algorithms

Several algorithms for mapping problem solution are presented in this section below.

5.3.1 An exact searching algorithm

An exact algorithm is based on a branch-bound method [Min86] and takes into account the peculiarities of the mapping problem by building the search tree, choosing the bounding function to be assigned with each leaf of this tree and searching for the “optimal” vertex on each step that corresponds to the component that has to be mapped.

Actually, the exact algorithm we propose, consists of the following steps:

1. Mapping the pre-attached components (that are assigned to particular computers by the client in advance).
2. Choosing the next component to be mapped.
3. Generating all acceptable locations for this component.
4. Computing the bounding function for every feasible location.
5. Choosing the best assignment for the component using the bounding values found above.
6. Checking whether all components have been assigned. Return to step 2 if there’re unassigned component in a DMA otherwise an optimal DMA placement in the DCS is found.
7. The obvious advantage of the exact mapping algorithm is concluded in its orientation on obtaining an optimal mapping variant. However, exponential complexity of its implementation does not allow using this method in practice for really large mapping problems.

Let us consider several rules that allow to improve time complexity of the algorithm (still keeping it as exponential).

The first problem we face at step 2 is to select the unassigned component. All selection rules can be divided into the two general classes: static and dynamic.

A static selection rule orders all components with respect to a chosen criterion only once, usually at the beginning of the algorithm. Then the algorithm chooses sequentially, from this ordered set, the next component to assign. An advantage of static rules is its low time expense. A disadvantage is the impossibility to take into account dynamic features of assignment process, unassigned part of the DMA topology and available DCS resources.

A dynamic rule computes a selection criterion on every step of an algorithm. Dynamic rules do not need a preliminary complete ordering of all components as the static mode. It is enough to find the next component according to criterion chosen without ordering the other components every time. A criterion of a dynamic rule can take into account dynamic features mentioned above; and therefore, in comparison with static one, usually it narrows the domain of optimal solution search more effectively. However, on the other hand, the regular re-computation of the criterion increases time expenses and the total algorithm complexity.

Before considering several rules to select a component, we would like to introduce so-called TIGHTNESS value. As computational experiments have shown, TIGHTNESS is very useful for algorithm efficiency analysis and to get a domain of advantageous usage for rules and heuristics.

TIGHTNESS value is used to specify the relative difference between the amount of available DCS resources and the amount of DCS resources required by the DMA components and links. Let D_i be the amount of a resource required by component i, and R_n be the amount of available resources on computer n. Then for computer n and for given resource type

$$TIGHTNESS(n) = \Sigma_i D_i / R_n * 100\%, \tag{5.6}$$

where the sum is taken over all components for which the computer n is acceptable.

Actually, when TIGTNESS value is set to 100, one can expect the redundancy of available network resources. That fact, in its turn, may lead to the existence of a large number of feasible mapping variants, and finding an optimal allocation in this case may require more time. With TIGHTNESS value set to 100, we can also expect that all DMA components would finally be allocated to the same DCS computer thereby making the communicational expenses negligible. However, this way doesn't always lead to the optimal solution, especially when the computational resources required by the DMA components excel the communicational expenses required by the DMA graph in values.

By decreasing the TIGHTNESS value, one can achieve the situation in which there exist only few possible DMA allocation variants over the DCS graph. Actually, there is a limit for the TIGHTNESS value below that we cannot obtain any feasible mapping variants.

Several examples to demonstrate the effect of TIGHTNESS on solutions found by different algorithms will be presented below.

5.3.1.1 Component selection rules

The proposed component selection rules and their characteristics are presented in Table 5.1.

The *first* simple one is to choose the next component in the increasing order of the number of acceptable computers. Thus, the component that has less initial acceptable locations will be chosen for assignment before others.

Obviously, in the dynamic mode, the minimum number of acceptable computers can only be decreased during the algorithm execution with increasing number components assigned and decreasing available DCS resources. If not yet assigned component i has a current set ζ_i of acceptable computers then the next component with minimum{$|\zeta_i|$, i is varied over all not yet assigned components} will be chosen, where $|\zeta_i|$ is a cardinality of set ζ_i .

This dynamic rule is very useful if, on every step of the algorithm, there exists only one acceptable computer for one or more components. Such components have so-called non-alternative assignment. Every such component must be assigned as early as possible. If its assignment will be postponed then chances to map the component will decrease in time because the available resources of the acceptable computer will decrease as well. This rule allows rejecting, as early as possible, the vertices in a search tree that can lead to a DMA allocation that is not acceptable. So, the rule supports the narrowing of the domain of optimal solution search.

Note, that the pre-attached components, that are assigned by the client in advance have also non-alternative assignments. At step 1, the algorithm checks the possibility of such assignments.

The rule to select the next component in the order of increasing the number of acceptable computers is one of the best from the point of time expense for static mode. However, in the static mode, only a situation at the beginning of the algorithm is considered. On the contrary, in the dynamic mode, current number of acceptable computers for each component is taken into account. That is right for all other rules: a dynamic rule takes into account a current value of a criterion used.

The *second* rule considered is to select the next component in the order of decreasing the resources required for component allocation. So, a component with largest resource demand will be chosen before others. This rule is useful in the case when one or more computers are acceptable for many components that have a large variation in the resource demands provided that the computers have a low or moderate TIGHTNESS. The rule allows to avoid the situation when components with relative low resource demands will be placed first on such computer and do not permit to place the components with higher resource demands.

Table 5.1 Rules for choosing the next component to be assigned

Rule	Static			Dynamic[1]		
Selecting the next not yet assigned component that has	Minimum number of computers acceptable for the component	Maximum resources required by the component	Maximum degree	Minimum number of computers acceptable for the component	Maximum resources required by the component	Maximum degree
Advantages	Low time expense	Low time expense	Low time expense	More effective for narrowing the solution search domain	More effective for narrowing the solution search domain	More effective for narrowing the solution search domain
Disadvanta-ges				High time expense	High time expense	High time expense
When the rule is useful	Large variation of acceptable computer number for different components High intersection of computer sets acceptable for different components	Low TIGHT-NESS Large variation of computer resource demands High intersection of acceptable computer sets	High TIGHT-NESS Large variation of component degrees There are computers with high acceptabilit y and resource availability			

Because there are more than one type of resources (e.g., CPU, memory, bus rate, interface bandwidth etc.), a transformation of component requirements for resources of different types to one scale can be made. Several different approaches to build up the criterion function can be proposed. Let us consider some of them.

Let RT_{nk} be the available DCS resource of type k on computer n, and D_{ik} be the type k resource demand of component i. Ratio D_{ik}/ RT_{nk} denotes the fraction of computer resource of type k required by component i if latter will be assigned to computer n. The optimistic and pessimistic criteria for component arrange can be defined in min-max and max-max forms correspondingly:

$$P1_i = min \{max [(D_{ik}/ RT_{nk}), \forall\ k], n \in \zeta_\iota \}$$
$$P2_i = max \{max [(D_{ik}/ RT_{nk}), \forall\ k], n \in \zeta_\iota \}$$

The optimistic (pessimistic) criterion means that component placement on the computer with minimum (maximum) required resources is expected.

[1] Dynamic rules do not need a preliminary complete ordering all components. It is enough to find the next component in accordance with the rule without ordering the other components every time.

As follows from the definitions of the criteria, firstly for given acceptable computer, the type of resource with maximum fraction of usage by the component (or, in other words, with minimum resource excess) is obtained. Then, for the optimistic case, minimum (or, for the pessimistic case, maximum) of obtained values over all computers acceptable for the component is found.

Of course, other criteria defined over component resource requirements and DCS resource availability can be specified. For example, following parameter $P3_i$ represents the resource demands relative to the total available DCS resources of acceptable computers

$$P3_i = \Sigma_k (D_{ik} / \Sigma_{n \in \zeta} RT_{nk}).$$

The *third* rule proposed is based on DMA topology peculiarities and takes into account so-called degree of a component. The degree means a number of links incident at the component. The degree shows the communication connectivity level of the component with the rest of DMA topology graph. So, the rule to select the component with highest degree allows placing first 'gravitation centers' of DMA topology. If such components will be placed on computers with high acceptability and resource availability, then it can be expected that minimum communication expenses will be achieved after DMA allocation.

In practice, a combination of the rules considered can be used in the mapping algorithm (e.g., first and second rules, or first and third rules). We propose to detect the peculiarities of the DMA and the DCS before use one particular or any combination of the rules. In Table 5.1 the conditions those favor the use of the individual rules are depicted. Further analysis of the rule computational efficiency based on experiments permits to refine such conditions.

The efficiency of proposed and other rules for component selection rules is for further research.

5.3.1.2 Bounding function

At each stage of the algorithm for every terminal vertex of the search tree, a lower bound of objective function F in (1) is needed. Suppose components of set Z are already assigned and ones of set Y are not yet assigned. The cost function F can be presented as sum of two term F = F(Z) + F(Y), where F(Z) and F(Y) are the allocation cost of components of set Z and Y respectively. Using formula (1), one can compute the placement cost F(Z) of assigned components. To compute a lower bound for F(Y), the best placement for every unassigned component and its not yet mapped links is determined and the total cost of such allocations is used for the lower bound

It's important to note that computations of the bounding values may be skipped in order to diminish the total complexity of the algorithm, to decrease the amount of memory required and to achieve the more compact implementation. However, in this case, the search tree may grow up unpredictably and therefore the total time for obtaining the optimal solution can increase considerably.

5.3.2 Approximate algorithm

In [IH96] efficient approximate mapping algorithm is proposed. It can allocate chain or tree structured DMA consisting of several heterogeneous components across the distributed computers interconnected using a general purpose LAN interconnection.

In this section, we present an approximate algorithm as a simple modification of the exact algorithm proposed above. It allows considering general DMA topology and does not constraint communication network structure. The complexity of the algorithm remains exponential but the time required for obtaining pseudo-optimal solution depends on a given acceptable error of the solution, and the time can be considerably less than the run time of the exact algorithm.

Suppose, one would be satisfied with the solution *A* produced by the mapping algorithm if it differed from the optimal solution *B* by no more than *eps*:

$A = B * (1 + eps')$, $/eps'/ \leq eps$

where *eps'* is the actual relative error of the solution produced by the approximate algorithm.

Further, to satisfy this constraint we propose to change the current search direction in the exact algorithm only if there exists another search direction in the search tree with the bounding value that exceeds (or is less than) the current one by *eps* or more. By going this way, we will obtain the solution that differs from the optimal one by no more than ***eps*.** The search tree will grow "in depth" faster and the amount of memory required as well as time required for the implementation will decrease. It's important to note that *eps* may be tuned interactively during the process of mapping. Depending on *eps* specified one can obtain less accurate results in shorter time or the solution very close to the optimal one but with greater time complexity (the more *eps* we specify the "faster" and less accurate results we'll get). For example, when the search tree grows "in breadth" rapidly thereby slowing the algorithm down, it may be justified to set the larger *eps* in order to make the tree grow "in depth".

Obviously, if $eps = \infty$ then so-called 'depth first search' method [Min86], which chooses a vertex of maximum depth among those vertices not yet branched. If there is more than one, then one could choose that which corresponds to the lowest bounding value. This method aims at exhibiting a (good) solution of the problem as soon as possible. Then value F_o of the got solution is used for narrowing the domain of optimal solution search by rejection of those vertices for which lower bound of the objective function is not less than F_o. By limiting, at each stage, the computation of the bounds to the successors of only those vertices for which lower bound is not more than F_o, a considerable reduction of the number of vertices actually examined can be obtained, a reduction sufficient to deal with problems of a fairly large size.

However, this modification won't work properly for large-dimensional mapping tasks just because one will be forced to set too large *eps* in order to produce the solution in acceptable time. It's clear, that the error of the results obtained in this way will depend on the dimension of a mapping task (e.g. on the number of DCS computers) and will be inadmissible for large mapping problems. That is why an effective heuristic method to find the pseudo-optimal solution with an acceptable error and an acceptable time complexity should be developed. In general, this is a problem to find just a feasible assignment of an arbitrary DMA graph over an arbitrary DCS structure.

5.3.3 Heuristic algorithm based on cluster analysis

As it was mentioned above, by mapping a DMA to a DCS one has to optimize the usage of communicational resources required by the DMA components connected via links. When assigned to the same computer, interacting DMA components need no additional resources to transfer the data. That is why the problem to optimize the usage of communicational resources in a DCS might be considered as a problem of grouping the most "interacting" DMA components on a single DCS computer thereby making possible communicational expenses negligible. Moreover, minimization of the communication channels used favors indirectly improvement of end-to-end QoS such as delay and reliability.

The optimization problem considered differs from the original one (5.1) – (5.5). So, the problem is to place a DMA in a DCS such that to minimize total communication resource (bandwidth) used satisfying resource constraint (5.3) and (5.4). In other words, the problem is to partition the given set of the DMA components into some N subsets (clusters), where N is not more than a number of DCS computers available for mapping, and to map each subset to an individual computer taking into account resource constraints.

We propose an approach based on clustering DMA components to solve that problem. Pre-attached components placed on computers in advance represent a set of assigned clusters. All pre-attached components placed on a same computer are merged in an individual cluster. Further, an assigned cluster can be enlarged by including new not yet assigned DMA components. Therefore the assigned clusters are taken into account as well as unassigned DMA components.

Let object denotes an unassigned DMA component, or assigned cluster. We merge objects sequentially. On every current step, we select two objects with maximum interconnection weight that must be merged next such that to minimize the cluster interconnections. (The mutual interconnection weight between DMA components will be defined below as a function of bandwidth required by a link connecting them). If both objects are unassigned DMA components then we try to include them, or at least one of them, in a new cluster that is not yet assigned. At every time we have only one new cluster. If one of two selected objects is a cluster already assigned then we try include not yet assigned object in that cluster. If both objects represent different clusters already assigned to different computers then the interconnection between such the objects is marked as disabled and we choose other pair of objects.

We assign every new cluster formed to a certain DCS computer chosen in advance according to a particular policy. We select a DCS computer with the maximum amount of resources currently available. This strategy is oriented on building larger clusters and minimizes the communication resources used.

We constrain the size of a new cluster formed by the amount of available resources on a DCS computer this cluster is pre-assigned. One can propose two approaches to take into account the resource constraints of the DCS computer. The first one, so-called pessimistic approach, assumes that all unassigned components adjacent at the new cluster will be placed on different computers, and it checks the resource constraints (3) - (5) both for components and for their sending and receiving links.

The second approach, so-called optimistic one, assumes that all adjacent unassigned components will be placed on a same computer and no resources for links will be needed. It means that only resource constraints (3) - (4) are checked and only for components and not for their sending and receiving links.

Obviously, the optimistic approach consumes less computational time than the pessimistic one but it obtains more often DMA assignment solutions that are not feasible because of a violation of resource constraints for ingoing and outgoing links.

On the other hand, the pessimistic approach guarantees that no such the violations occurs but sometimes it can not find a feasible solution that exists due to the assumption it makes. Actually, it can reject a component assignment because the available I/O bandwidth is not sufficient to satisfy the required total bandwidth of all incident links to communicate with adjacent not yet assigned components. However some of such the components could be placed on next steps on the same computer and, therefore, do not really need the communication resources.

Let us introduce factor P which denotes "pessimism-factor" - the less it is the more optimistic the approach will be. In case P = 1 we get absolutely pessimistic approach, in case P = 0 we get the optimistic one. Let B be the bandwidth required by links not yet mapped and incident at a component that must be assigned next. Then one compares B*P obtained with the target computer’s I/O capability and decides whether the new component can be included in the cluster or not.

Let New_Cluster be the currently building cluster not yet assigned. Let Target be a DCS computer the New_Cluster is assigned to.

Then the implementation of suggested algorithm is as follows:

1. Mapping the pre-attached components. If any resource constraint is violated then no acceptable DMA location in the DCS exists and go to 15.
   All components assigned to a same computer are merged in one cluster. The assigned clusters are taken into account on next steps as well as unassigned DMA components.
2. Mark the interconnections between assigned clusters as disabled. If any cluster has no (allowable) interconnections with not yet assigned DMA components then exclude such the cluster from further considertation.
3. Create empty New_Cluster.
4. Using CLUSTERING procedure, find the next two objects to group. Mark the found objects as Obj1 and Obj2.
5. If one of objects Obj1 and Obj2 is a cluster already mapped to computer n then check whether the computer n has enough resources to place the unassigned object as well. If no then{mark the interconnection between these objects as disabled; if the object already assigned has no interconnections with other objects then exclude such the object from further consideration; return to step 4} otherwise merge both objects in one cluster and place this cluster on the computer n.
6. If New_Cluster is not empty then go to 10. (Both objects Obj1 and Obj2 are unassigned DMA components or one of them can be New_Cluster that is not yet assigned as well).
7. Add Obj1 to New_Cluster, if Obj1 had not been already included in New_Cluster.

8. Map all non-alternative DMA components that have a single acceptable computer on current step. If any resource constraint is violated then no acceptable DMA location in the DCS is found and go to 15.
9. If no assigned clusters exist then using maximum available resources strategy, otherwise using maximum allocated components strategy, find Target that is acceptable for both Obj1 and Obj2. If such the Target is found then go to 11 otherwise mark the interconnection between Obj1 and Obj2 as disabled and select another Target that is acceptable for New_Cluster (i.e. Obj1). If it is successful then go to 11 otherwise go to 15.
10. Add Obj1 to New_Cluster, if Obj1 had not been already included in New_Cluster.
11. If Target cannot hold New_Cluster due to lack of available resources then delete the last added object from New_Cluster and allocate New_Cluster on Target.

    Mark the interconnections between New_Cluster and other clusters (which are assigned already) as disabled. If any cluster has no (allowable) interconnections with not yet assigned DMA components then exclude such the cluster from the further considertation
12. If all DMA components are assigned to the DCS computers then go to 14, else go to 3.
13. The same as steps 10, 11 and 12 but with Obj2
14. DMA assignment was completed successfully. Stop.
15. DMA assignment failed. Stop.

Mentioned optimistic and pessimistic approaches for checking resource constraints (that influence the Cluster size) are used at steps 1, 5, 8, 11 and 13 of the algorithm. At step 1, if pessimism-factor $P>0$ then I/O constraints are checked for ingoing and outgoing links connecting only preattached components and for the links (preattached component, non-preattached one) where non-preattached component has not preattached component's computer in ist acceptable computer list or can not be placed on that computer because of resource constraint violation.

The procedure of adding an object to a cluster includes the object in the cluster and updates the DMA graph such that the node representing that object and links connecting it with the Cluster are removed, and the node representing the cluster gets additional links that have communicated the object included with its adjacent ones that are not in the Cluster. If a pair of duplicated links is obtained, then it must be replaced by one link with total bandwidth weight equal to sum of two previous ones.

Let D be maximum degree of a component in the DMA graphs, N be the number of unassigned DMA components, M be the number of computers. Then the worth case algorithm complexity would be bounded by O(ND) for steps 1 and 8, $O(KN^3)$ for step 3, O(M) for step 9, O(N) for steps 4, 5 and 7, and constant for others. Here K is the number of so-called interconnection matrix transformations (see below). As shown in [Sol91], for practice purpose, value of K not more than 10 is usually sufficient. Thus, the worth case complexity for every cycle of the algorithm is bounded by $O(aKN^3+ bM)$, where a and b are constants. Number of the cycles is not more than the number of components that must be clustered and assigned to computers. Then the total complexity of the algorithm is bounded by $O(aKN^4+ bNM)$ in the worst case . Really, the complexity is less because the number of unassigned (non-clustered) components decreases from N to 1. Therefore more correctly, the algorithm complexity is bounded by $\Sigma_{n=1,N}$ $O(aKn^3+ bM)$.

5.3.4 An algorithm to find the feasible assignment of a DMA to a DCS

Several approaches can be proposed to find a feasible assignment of a DMA to a DCS.

As follows from 5.3.2, such an assignment can be produced by the approximate method with *eps* set large enough or assigned to ∞.

Of course, a DMA assignment that is found by the heuristic algorithm (see 5.3.3) added by the complete checking resource constraints for such the assignment can be considered as a feasible placement also.

Below we propose yet another algorithm that is a modification of exact algorithm and is characterized by following features:

- No bounding values for vertices are computed.
- The 'depth first search' method mentioned above is used for the branching vertex at each stage.

Before presenting the algorithm, we remember the procedure of the search tree construction. As known [Min86], the search tree is structured hierarchically in such a way that every level of the tree corresponds to individual component, and vertices of a level correspond to acceptable locations of the component. A rule of the vertex selection at a particular level of the search tree determines what computer is chosen for assignment of the component corresponding to such the level.

The algorithm we propose, consists of the following steps:

1. Mapping the pre-attached components
2. Choosing the next component to be mapped
3. Generating all acceptable locations for this component.
4. If there is no acceptable location and the component is the first one then there exists no acceptable DMA location in the DCS
5. If there is no acceptable location and the component is not the first one then return to the previous component assigned (i.e., to the previous level of the search tree), exclude its assignment from the further consideration, set the previous component as current one, and go to step 4
6. Choosing the acceptable location for the component
7. Checking whether all components have been assigned. If no then return to step 2 else a feasible DMA placement in the DCS is found.

Skipping the bounding value computations conduces to reduction of the algorithm time complexity if and only if a rule of the selecting an acceptable computer for the component permits to construct the search tree mainly in vertical direction (i.e. in depth) and to avoid the growing the search tree in horizontal direction. Note that the component selection rules mentioned above in 5.3.1.1 can be used at step 2 of the algorithm. And it is important to agree the component selection rule with component allocation selection rule. For example if the second rule of maximum resources required by a component is used for choosing the next component (see Table 5.1) then, to select a location for the component, the rule of maximum available resources of the computer have to be used.

The proposed component location selection rules and their characteristics are presented in Table 5.2.

5.3.5 Initial asSIGnment iMproving Algorithm (SIGMA)

In this section, we consider iterative algorithms that start with a feasible initial assignment of DMA components and try to improve the DMA allocation from the point of objective function. Such an assignment can be found in advance using one of the algorithms of sections 5.3.3, 5.3.4

5.3.5.1 Original algorithm

In [HDR96a], such the algorithm is proposed. It is based on so-called sequential method [Sol91] that is characterized by polynomial complexity. However the complexity of the proposed algorithm depends not only on the sequential method but also on the procedure used for component removal from one computer to another so that a new DMA allocation obtained will be feasible as before. No effective algorithm for such the procedure was proposed.

Besides, the algorithm uses the notion of so-called redundant computer. For a current acceptable DMA allocation, a computer is defined as redundant one if another acceptable DMA allocation can be obtained without this computer and with objective function value less than current one. It was shown that use of a redundant computer does not permit to get optimal solution. A heuristic criterion to detect redundant computers was proposed in [HDR96a]; but it can not guarantee that the algorithm will exclude all redundant computers from the consideration.

The idea of the algorithm is as follows. At the beginning, all DMA components, excepting pre-attached ones, are marked as *black* components. A black component is a component that is allocated in any way into the DCS but the algorithm does not yet confirm its allocation. So, each component starts with a given (black) placement and ends up with a confirmed (so-called white) placement. In between, there can be at most one reallocation for each component. Thus, at each step the algorithm either removes any black component to a new location or confirms the current location of any black component. In the latter case the black component becomes white and after that it can not be removed at all.

Below we propose a heuristic modification of the algorithm that, in comparison with original one, does not use the conception of redundant computers, realizes the removal procedure in polynomial time, and therefore is characterized by a polynomial complexity.

Table 5.2 Rules for choosing component location

Rule	Static				Dynamic		
To place the compo-nent, select the computer that has	Maxi-mum number of compo-nents for which the computer is accept-able	Maximum available CPU resource	Maximum available communi-cation resource of its interfaces	Preference in round-robin discipline	Maximum number of componen ts, for which the computer is acceptable	Maximum available CPU resource	Maximum available communi-cation resource of its interfaces
Advanta-ges	Low time expense	Low time expense	Low time expense	Low time expense	More effective for narrowing the solution search domain	More effective for narrowing the solution search domain	More effective for narrowing the solution search domain
Disad-vantages					High time expense	High time expense	High time expense
When the rule is useful	Large variation of accept-able compo-nent number for different compu-ers High intersec-tion of compo-nent sets accept-able for different compu-ters	High TIGHT-NESS Large variation of available CPU resource for different compo-nents	Low TIGHT-NESS Large variation of available interface communi-cation resource for different computers High intersec-tion of acceptable computer sets	High TIGHT-NESS but less than 100% There are computers with high accept-ability			

5.3.5.2 *Modified algorithm*

Let us first describe some concepts particular for the algorithm.

Firstly, we construct so-called cost matrix C. It consists of N rows (N is a number of the black DMA components) and M columns (M is a number of the acceptable DCS computers). Each element c_{in} of the matrix represents the "local" cost of mapping DMA component to computer n. At every stage of the algorithm there exists a current acceptable DMA allocation, i.e. all components are placed, therefore the cost of any replacement of one component can be exactly computed. Moreover, the procedure of computing elements of the matrix can take into account the peculiarities of the DMA and the DCS mentioned above in 5.1, namely multicasting and dependence of component allocation cost on placement of adjacent components.

If component i can be allocated on computer n then cost elements c_{in} is computed, using cost matrices f, u and g for cost evaluation of mapping components, outgoing and incoming links to computers and links to virtual channels correspondingly (see 5.2)

$$c_{in} = f_{in} + (\Sigma_j\ u_{(ij)(nm)1} + u_{(ji)(mn)2}) + (\Sigma_j\ g_{(ij)(nm)} + g_{(ji)(mn)}) / 2.$$

Here the first term is the cost of mapping the component i to computer n. The first sum is the total cost of mapping sending side of all outgoing links and receiving side of all incoming links of component i to computer n. The second sum is the total cost of mapping output and input links incident at component i to corresponding output and input channels of computer n. Obviously, the mapping of the links depends on the allocations of the adjacent components, i.e. on the current DMA allocation in the DCS. The factor ½ is used to share the cost for the communication channel between component i and the other one, connected to it via the link.

If component n is not acceptable for allocation of component i then we assume $c_{in} = \infty$.

The cost matrix C is transformed to the so-called INTERCONNECTION_MATRIX H^0 [HDR95] according to the formula

$$h^0_{in} = max\ \{c_{kl}, \forall\ k,l\} - c_{in}$$

Matrix H^0 provides transform the original minimization problem (5.1) – (5.5) to a corresponding maximization problem. Element h_{in} of the matrix H^0 can be interpreted as an economic benefit if component i will be assigned to computer n.

Secondly, we build so-called relative INTERCONNECTION_MATRIX H^1 in other way as described in [HDR96a], to find the next assignment (component, computer).

To avoid the influence of the redundant computers (see [HDR96a]), we propose the following formula for the element of matrix H^1:

$$h^1_{in} = h^0_{in} / \Sigma_k h^0_{ik} \qquad (5.7)$$

where h^1_{in} is an element of the matrix H^1. In accordance with (5.7), element h^1_{in} evaluates a relative gain of the assignment of component i to computer n relative to other computers acceptable. Maximum element h^1_{in} of R determines next assignment i to n to be made.

In comparison with the algorithm [HDR96a], the transformation of the INTERCONNECTION_MATRIX by (5.7) is now made only once because, after the first transformation, all elements will be not change and the sum of elements of any row will be always equal to 1.

According to the formula (5.7), the search of redundant computers is not needed any more. Actually, algorithm proposed in [HDR96a] and based on the sequential method [Sol91] uses multiple transformation of matrix H^0 by the recurrent formula

$$h_{in}(k) = h_{in}(k-1) / (\Sigma_l\ h_{ln}(k-1) + \Sigma_l\ h_{il}(k-1)),\ \forall\ i,j;\ k > 0 \qquad (5.8)$$

where $h_{ij}(k)$ is an element of the matrix H^k of the k-th order.

As follows from (5.8), two kinds of interconnections between components and computers are taken into account. The first kind of interconnections reflected by the first sum in (5.8) and represented in (5.7) is a gain factor of the assignment of *given component i* to a computer n *relative to other computers acceptable*. So, it permits to find the best location for given component.

The second kind of interconnections reflected by the second sum in (5.8) is gain factor of selecting component i to assign it *to given computer n relative to other components* for which the computer n is acceptable too. So, it allows to find what component can satisfy the given computer n in the best way.

Thus, formula (5.8) proceeds from the fact that every component must be allocated and every computer has to be used for the DMA allocation. In other words, every computer has to get at least one DMA component. However the last premise is not right for the DMA mapping problem as shown in [HDR96a]. Just that is why the concept of redundant computers was introduced in the original algorithm proposed in [HDR96a].

Note that formula (5.8) was successfully used in the sequential method [Sol91] to solve (in polynomial time) the classical assignment problem of N individuals and N jobs. In such the problem, really, each individual must be assigned to a job and each job has to get an individual.

Contrary, formula (5.7) takes into account only “the interest of components to be allocated in the best way” and ignores “the interest of every computer to catch the best (in the sense of the computer) DMA component”. Therefore formula (5.7) does not need the concept of redundant computers.

Thirdly, we introduce the parameter LAST_ACTION to determine the last action taken place thus simplifying the algorithm implementation. These actions are:

LA_START that refers to the beginning of the algorithm

LA_ENABLEBLACK which means that a DMA component has been re-mapped for the first time onto a DCS computer and remains “black” (see [HDR96a])

LA_ENABLEWHITE which means that a DMA component has been mapped onto a same DCS node for the second time thereby becoming “white”

LA_DISABLED which means that a pair (component, computer) has been disabled (see [HDR96a])

The algorithm proposed is as follows:

1. Set LAST_ACTION to LA_START
2. If (LAST_ACTION equals to LA_ENABLEBLACK) compute COST_MATRIX for the current DMA assignment
3. Transform COST_MATRIX to the INTERCONNECTION_MATRIX H^0 and the latter to the relative INTERCONNECTION_MATRIX H^1 by the single matrix transformation
4. Choose the maximum element (i,n) in matrix H^1
5. If component i is already allocated on the computer n then mark i as a white component, exclude it from the COST_MATRIX, set LAST_ACTION to LA_ENABLEWHITE and go to 9
6. If component i can be placed on computer n without resource violation then mark i as a black component, set LAST_ACTION to LA_ENABLEBLACK and go to 9
7. Try to remove the black components from the computer n until its available capacity will be enough to allocate the component i. If the removal is successful, go to 6.
8. Mark pair (i,n) as disabled in the COST_MATRIX and matrices H^0 and H^1, set LAST_ACTION to LA_DISABLED and go to 2
9. If there are no black components then the best DMA assignment has been found, stop, else - go to 2

Component REMOVAL procedure described in [HDR96a] is now implemented in the heuristic way to achieve the faster implementation. Components can be moved only one by one (grouped components movements are not considered) so that each component can be assigned directly to the determined destination computer without multiple movements.

The proposed matrix transformation and removal procedures provide a reduction of the algorithm complexity and its running time.

5.3.5.3 Complexity of the algorithm

For every step of the algorithm proposed, the worth case complexity would be bounded by

1. Constant

2. O(DNM).
3. $O(NM^2)$
4. O(NM)
5. O(NM)
6. O(D)
7. O(DNM).
8. Constant

So, the complexity of one algorithm cycle in the worst case is bounded by $O(DNM^2)$. In the algorithm the number of removal of each component is restricted by one. Therefore the number of the cycles is not more than twice the number of components that must be assigned to computers. Then the total complexity of the algorithm is bounded by $O(DN^2M^2)$ in the worst case . Really, the complexity is less because the number of unassigned components decreases from N to 1. Therefore more correctly, the algorithm complexity is bounded by $\Sigma_{n=1,N}\ O(DN^2M^2)$.

5.4 Experiment: computational complexity and accuracy of mapping algorithms

5.4.1 Technology

We have tested the algorithms we proposed on the large number of numeric examples. Each test we made corresponds to a randomly weighted DMA graph being mapped to a DCS structure with random parameters. All DMA graphs were generated in advance and every graph we used had a certain base structure that was replicated by a given factor to produce the needful testable graph. Examples of testable graphs are shown below.

5.4.2 Testable DMA graphs

We have considered two different types of three-level and a type of five-level graphs that are typical for distributed multimedia applications. Such kinds of graph structures we have used as the examples to test the algorithms we proposed. The DCS structure was formed by a variable set of computers connected via LAN. We have assumed that source and sink components are assigned to computers in advance and the problem is to find the optimal placement for intermediate components of a DMA structure.

5.4.2.1 The first testable DMA graph

The first graph we considered (GRAPH1) was being generated from the base one depicted in Figure 5.1 by replicating the base structure by a given factor. For example, GRAPH1(3) is depicted in Figure 5.2 and corresponds to the factor equal to 3. Thus, the number of components that have to be assigned is equal to the replicated factor n.

In general GRAPH1(n) consists of the three levels: the level of source components (depicted in gray), the level of mixers (depicted in white), the level of sink components (depicted in black).

Source components generate multimedia data, mixers process and transform this data, and sinks consume the incoming streams.

Other DMA graph parameters (such as component CPU and memory requirements) were generated in a random way.

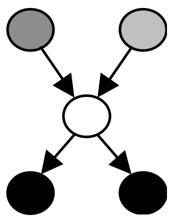


Figure 5.1 The base graph 1

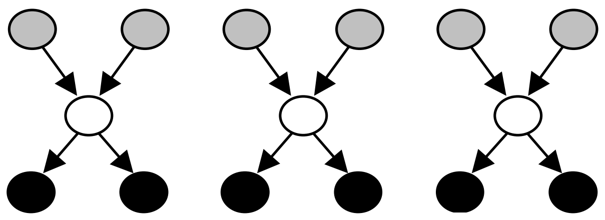


Figure 5.2 The testable three-level graph GRAPH1(3)

5.4.2.2 The second testable DMA graph

The second graph we examined consists of the three components levels as well - the "source" level, the level of mixers and the "sink" level. This graph - GRAPH2(n, m) was parameterized by the number of source components - n and the number of mixer components - m (the number of sink components was supposed to be the same as for sources only to restrict the variety of such graphs). Every source component is connected with every mixer which, in its turn, is connected with every sink.

The number of mixers m defines the components that have to be assigned[1].

An example of such the DMA topology is CSCW, where n users interaction with each other in m shared spaces. Such spaces could be shared whiteboard, virtual space, audio conference (audio space), etc. For example, each user has one source and one sink component for every space type. One mixer, m = 1, (corresponding to a space) receives the data flows from the sources of the users, mixes the data and transforms the mixed data to sinks of the users.

5.4.2.3 The third testable graph

The third graph GRAPH3(n,m) we considered consists of the five following components levels: source level with n source components, level of n filters connected to sources (shortly, source filter level), mixing level with one mixer, sink filter level with m filters and sink level consisting of m sink components. The example of graph GRAPH(3,2) is depicted in Figure 5.5. Testable graphs

[1] The number of sources (sinks) n is considered also as parameter of this type of testable graph because, in experiments conducted, the number of acceptable computers is defined as total number of DMA components, i.e. 2n+m. Thus parameter n also influences on the mapping problem dimension.

GRAPH3(n,m) are generated by replicating the components of source and sink levels and both filter levels corresponding to two previous ones.

For this kind of testable graphs, the number of free components for assignment procedure is equal to (n+m) + 1.

Multimedia bridge in Figure 5.5 is likely an intellectual mixer responsible for computing various multimedia streams sent from one network to another one with different communication protocol parameters.

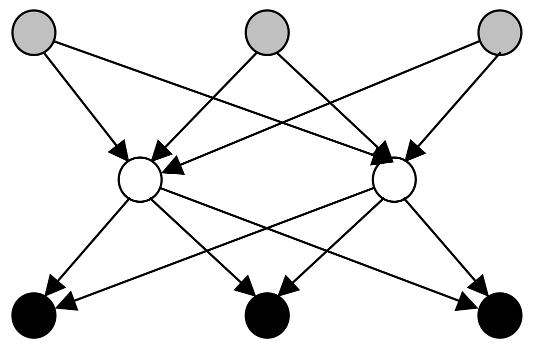


Figure 5.3 The testable three-level graph GRAPH2(3,2)

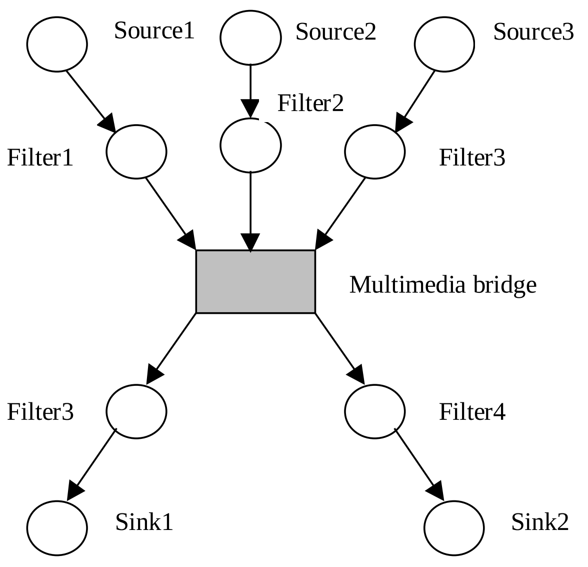


Figure 5.5 The testable five-level graph GRAPH3(3,2)

5.4.3 Results

For each DMA graph we considered, experiment was made once, and results for every experiment are represented in a row of the tables shown below.

We have obtained for different testable DMA graphs described above:

- the financial cost of the solutions produced by the different mapping algorithms proposed (see 5.3),
- the cost of the initial assignment produced by one of constructive algorithms (e.g. by one of the algorithms proposed in 5.3.4 or by the heuristic algorithm suggested by G.Dermler) and the improved assignment obtained by the iterative algorithm SIGMA (see 5.3.5),
- computational complexities of different algorithms,
- the relative cost differences between the optimal assignment produced by the exact algorithm (see 5.3.1) and the one produced by an algorithm analysed.

We have controlled:

- the number of DMA components (i.e. parameters of testable graphs),
- the number of DCS computers. For GRAPH1(n), this number was equal to 5n; for GRAPH2(n, m), it was equal to 2n + m, i.e. was the same as the number of DMA components; for GRAPH3(n) it was variable,
- the TIGNTNESS value.

As mentioned above, we used a TIGHTNESS value to control the relative difference between the amount of available DCS resources and the amount of DCS resources required by the DMA components and links. The TIGHTNESS value was varied from 10 to 100 % and it was the same for different computers and types of resources (CPU, memory, and bandwidth) in each experiment.

With TIGHTNESS value set to 100%, we can also expect that all DMA components would finally be allocated to the same DCS computer thereby making the communicational expenses negligible. However, this way doesn’t always lead to the optimal solution, especially when the expenses of computational resources required by the DMA components excel the communicational expenses required by the DMA graph in values. For example, in one experiment, the optimal solution found by the exact algorithm for the GRAPH1(3) differs from the solution produced by SIGMA, in which all DMA components had been allocated to the same DCS node.

By decreasing the TIGHTNESS value, one can achieve the situation in which there exist only few possible DMA allocation variants over the DCS graph. Actually, there is a limit for the TIGHTNESS value below that we cannot obtain any feasible mapping variants. For example, as depicted in Table 5.11, GRAPH2(6,5) had such a limit equal to 22%.

Results obtained for testable graphs are given below.

5.4.3.1 DMA graph GRAPH1(n)

In Tables 5.3 - 5.5, accuracy analysis of the iterative SIGMA algorithm is represented for the DMA graph GRAPH1(n) with different replicated factors n = 3, 4, 5 and for full-connected DCS graph with different numbers of computers 15, 20, 25 correspondingly.

The TIGHTNESS value was varied from 50% to 100%.

To obtain a feasible DMA initial assignment, the heuristic algorithm proposed by G.Dermler. This algorithm looks for DMA allocation that minimizes end-to-end delay for DMA structures restricted by mixing and multicasting zones. Therefore the evaluation of this algorithm errors displayed in the tables are used not for accuracy comparison with the exact solution but only to understand how close can be the solutions obtained if different criteria are used by the algorithms. Here the exact algorithm uses the cost criterion, and the algorithm for the initial feasible DMA placement – the end-to-end delay criterion.

The solutions obtained by the exact algorithm (see 5.3.1) are depicted in column ‘Exact’, initial DMA allocations produced by the Dermler’s algorithm are represented in column ‘Initial’, improved DMA assignments obtained by the SIGMA algorithm (see 5.3.5) are illustrated in column SIGMA. For every heuristic algorithm, the errors of solutions found are depicted also.

Accuracy analysis of SIGMA shows efficiency of the method we proposed - the relative error didn’t exceed 3.1% for all conducted experiments. Moreover, the computational time required by SIGMA, implemented in Unix/Solaris, SunSparc20, was shorter than 1s for all DMA graphs GRAPH1(n) and GRAPH2(n,m) we examined.

Table 5.3 Accuracy analysis of SIGMA algorithm for replicated factor n = 3

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	97 009	105 168	8.41	97 470	0.4
60	79 396	89 269	12.43	79 745	0.4
70	82 613	104 006	25.89	83 854	1.5
80	116 379	134 935	15.94	116 379	0
90	96 483	110 892	14.93	98 800	2.4
100	113 623	135 374	19.14	114 080	0.4

Table 5.4 Accuracy analysis of SIGMA algorithm for replicated factor n = 4

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	170 652	205 322	20.31	170 652	0
60	111 597	144 058	24.62	113 540	1.7
70	127 425	163 701	28.46	131 409	3.1
80	117 941	149 375	26.65	117 941	0
90	126 022	171 760	36.29	128 725	2.0
100	140 348	168 135	19.79	140 348	0

Table 5.5 Accuracy analysis of SIGMA algorithm for replicated factor n = 5

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	171 405	209 966	22.49	175 848	2.5
60	144 286	195 282	35.43	145 070	0.5
70	161 563	211 611	30.97	162 012	0.03
80	150 072	189 195	26.06	151 197	0.7
90	134 664	188 679	40.11	134 664	0
100	154 880	191 353	23.54	154 880	0

In Table 5.6, results of the accuracy and time complexity analysis of the cluster analysis based algorithm (shortly, cluster algorithm) and SIGMA algorithm are displayed for 20 DCS computers and DMA graph GRAPH1(n) with factor n = 4. Thus $(5n)^n = 20^4$ possible DMA assignments are possible, where 5n defines the computer number and n – number of components that must be assigned.

The cluster algorithm version represented in Table 5.6 uses the optimistic policy for target computer selection (see 5.3.3).

The computation time of the algorithms is measured for its implementations on Pentium 166 MHz 32 MB RAM, Windows 95.

It is important to note that the optimization criteria are the same only for the exact and SIGMA algorithms (that is cost function). For the cluster algorithm it is minimum total bandwidth of communication network used, and for the initial DMA assignment algorithm it is minimum end-to-end delay. Therefore, only SIGMA algorithm errors can be considered as accuracy algorithm estimation. The errors of other two algorithms only illustrate the closeness of solutions obtained if different criteria are used for DMA mapping.

In the columns of the cluster algorithm, the empty cells denote that this algorithm obtained no feasible solution when the optimistic strategy for the target computer selection is used.

Note that a combination of SIGMA with one of constructive algorithms (to obtain an initial DMA assignment) requires the computation time considerably less than the exact algorithm (namely, by ten times and more). It suggests that such the algorithms would be effective for real protocols in distributed multimedia systems.

Table 5.6 Accuracy and time complexity analysis of cluster and SIGMA algorithms for replicated factor n = 4 and 20 DCS computers

TIGHTNESS	Exact	Time ms	Cluster (optimistic approach)	Time ms	Error, %	Initial (Dermler's algorithm)	Time ms	Error, %	SIGMA	Time ms	Error, %
50	119 336	14 532	-	248	-	132 332	224	10.89	120 880	139	1.29
60	130 461	3 171	-	234	-	159 974	220	22.62	130 461	137	0
70	104 985	3 625	121 679	280	15.90	136 477	220	30.00	104 985	114	0
80	142 839	12 075	154 568	265	8.21	157 837	170	10.50	143 175	165	0.24
90	130 516	6 662	150 604	230	15.39	154 181	160	18.13	130 925	124	0.31
100	105 038	6 031	122 931	263	17.03	138 543	110	31.89	105 038	68	0

5.4.3.2 DMA graph GRAPH2(n, m)

In Tables 5.7 - 5.11 accuracy analysis of the iterative SIGMA algorithm is represented for the DMA graph GRAPH2(n,m) with variable number of DMA source components n = 2, 3, 4, 5, 6 and number of mixers m = 5. The number of DCS computers is defined as 2n + m and equal to 9, 11, 13, 15, 17 according to n.

The relative error of SIGMA does not exceed 6.3% for all conducted experiments.

The experiments conducted show that SIGMA is capable to reduce the large solution error of an initial DMA assignment to negligible value. For example, as follows from Table 5.7 for TIGHTNESS value 100, SIGMA could reduce the initial solution error 128% to 0% by re-mapping the DMA, i.e. SIGMA obtained the exact solution of the mapping problem.

Table 5.7 Accuracy analysis of SIGMA algorithm for n = 2 and m = 5

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	83 922	103 043	22.78	87 941	4.7
60	74 449	82 011	10.15	76 392	2.5
70	101 903	129 099	26.68	103 126	1.2
80	71 823	97 266	35.42	71 823	0
90	56 646	65 140	14.99	57 129	0.8
100	68 224	155 857	128.44	68 224	0

Table 5.8 Accuracy analysis of SIGMA algorithm for n = 3 and m = 5

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	105 933	170 291	60.75	11 607	5.3
60	96 110	152 530	58.70	100 104	4.1
70	107 988	126 673	17.30	114 811	6.3
80	109 838	114 055	3.83	11 429	1.4
90	118 667	144 823	22.04	119 530	0.7
100	111 625	152 880	36.95	111 812	0.01

Table 5.9 Accuracy analysis of SIGMA algorithm for n = 4 and m = 5

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	148 288	162 266	9.42	152 522	2.8
60	145 806	169 358	16.15	148 938	2.1
70	165 070	229 554	39.06	174 112	5.4
80	146 014	182 836	25.21	146 014	0
90	131 469	133 668	1.67	131 469	0
100	113 258	127 527	12.59	113 730	0.4

Table 5.10 Accuracy analysis of SIGMA algorithm for n = 5 and m = 5

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
50	204 331	209 389	2.47	208 470	1.9
60	189 477	272 457	43.79	189 477	0
70	175 591	242 703	38.22	175 591	0
80	162 033	179 545	10.80	162 630	0.3
90	155 091	231 031	48.96	161 451	4.1
100	146 598	203 509	38.82	146 598	0

Table 5.11 Accuracy analysis of SIGMA algorithm for n = 6 and m = 5

TIGHTNESS	Exact	Initial (Dermler's algorithm)	Error, %	SIGMA	Error, %
22	203 691	203 691	0	203 691	0
23	291 389	297 706	2.16	297 706	2.1
24	250 907	253 056	0.85	253 056	1.1
25	255 891	263 990	3.16	263 990	3.1
26	289 246	322 342	11.44	294 141	1.6
27	243 134	263 113	8.21	245 497	0.9
28	234 184	242 125	3.39	236 970	1.1
29	218 721	264 557	20.95	218 721	0
30	218 376	281 154	28.74	221 683	1.5
40	212 833	256 184	20.36	222 164	2
50	222 664	330 005	48.20	227 814	2.3
60	228 991	276 119	20.58	228 991	0
70	198 885	212 640	6.91	198 885	0
80	207 218	315 647	52.32	213 557	3
90	211 166	217 767	3.12	213 622	1.1
100	181 658	298 928	64.55	181 658	0

In Table 5.12, results of the accuracy and time complexity analysis of the cluster algorithm and SIGMA algorithm are displayed for 12 DCS computers and DMA graph GRAPH2(n,m) with n = 4 and m = 4. Thus $(2n + m)^m = 12^4$ possible DMA assignments are possible.

The computation time of the algorithms is measured for its implementations on Pentium 166 MHz 32 MB RAM, Windows 95.

The cluster algorithm obtained two DMA assignments that are not feasible because of the communication constraints violation.

Note that a combination of SIGMA with one of constructive algorithm requires again the computation time considerably less than the exact algorithm.

Table 5.12 Accuracy and time complexity analysis of cluster and SIGMA algorithms for n = 4, m=4 and 12 DCS computers

TIGHTNESS	Exact	Time	Cluster (optimistic approach)	Time	Error, %	Initial (Dermler's algorithm)	Time	Error, %	SIGMA	Time	Error, %
50	119 702	11 783	-	137	-	155 428	110	29.85	119 702	135	0
60	111 100	11 617	-	96	-	150 818	110	35.75	111 322	82	0.20
70	126 554	3 764	177 963	89	40.62	178 883	170	41.35	127 537	68	0.78
80	62 569	2 651	95 465	109	52.58	94 155	160	50.48	63 156	80	0.94
90	137 564	19 731	167 181	101	21.53	174 798	110	27.07	138 598	89	0.75
100	83 768	2 582	128 874	81	53.85	83 768	160	0	83 768	68	0

5.4.3.3 DMA graph GRAPH3(n,m)

In Tables 5.13 and 5.14, the accuracy and computation complexity analysis of the iterative SIGMA algorithm is represented for the five-level DMA graph GRAPH3(n,m) with variable numbers of sources and sinks. For (n,m) = (3,3), the mapping algorithm starts with 7 unassigned DMA components and 13 acceptable DCS computers; there are 13^7 possible DMA allocations in the DCS. For (n,m) = (4,4), there are 9 DMA components that must be assigned, 17 acceptable DCS computers and 17^9 possible DMA allocations.

To produce an initial feasible DMA assignment, the algorithm described in 5.3.4 was used. The static strategies were used to select the next unassigned component and the next acceptable computer. The components and computers were selected according to their numeration that was made arbitrarily in advance.

The computation time of the algorithms is measured for its implementations on Pentium 166 MHz 32 MB RAM, Windows 95.

Table 5.13 Accuracy and time complexity analysis of SIGMA for n = 3, m=3 and 13 DCS computers

TIGHTNESS	Exact	Time, ms	Initial	Time, ms	Error, %	SIGMA	Time, ms	Error, %
50	540	3 558	590	5 367	9.26	553	310	2.41
60	561	3 551	561	4 470	0	561	215	0
70	341	3 758	387	4 070	13.49	348	212	2.05
80	266	2 565	292	4 087	9.77	272	219	2.26
90	210	4 165	264	5 074	25.71	220	108	4.76
100	167	5 068	192	6 085	14.97	172	210	2.99

Table 5.14 Accuracy and time complexity analysis of SIGMA for n = 4, m=4 and 17 DCS computers

TIGHTNESS	Exact	Time, ms	Initial	Time, ms	Error, %	SIGMA	Time, ms	Error, %
50	735	15 020	850	34 510	15.65	750	410	2.04
60	860	29 651	910	32 070	5.81	871	523	1.28
70	750	40 100	855	85 704	14.00	772	109	2.93
80	634	60 625	649	80 200	2.37	649	619	2.37
90	920	63 165	985	> 100 000	7.06	932	650	1.30
100	467	140 268	543	> 100 000	16.27	475	1028	1.7

The relative error of SIGMA does not exceed 4.8% for all conducted experiments.

As follows from Table 5.13 and 5.14 the computation time of the exact and initial assignment algorithms depends on TIGHTNESS value, and SIGMA does not demonstrate such kind of dependency. Obviously, increasing the TIGHTNESS value increases the number of components that can be assigned to each computer, and, hence, increases exponentially the number of possible DMA assignments in the DCS. The size of the set, which is checked by the exact (or initial) algorithm, depends also on efficiency of the component (and computer) selection strategy used. Moreover, the computer with high TIGHTNESS value (close to 100%) and cheapest resources significantly narrows the domain of optimal solution search by assigning all (or nearly all) components to the computer.

Thus, the increasing TIGHTNESS value causes two contrary factors: exponential increasing the number of possible DMA assignments and narrowing the domain of optimal solution search because of possibility to assign most or all components to the cheapest computers with higher TIGHTNESS value. Beginning from Table 5.13 the first factor dominates that causes the sharp increase of computation time.

In contrary, computation complexity of SIGMA is a polynomial function of the number of unassigned components and the number of acceptable computers (see 5.3.5.3). TIGHTNESS influences only on element values of cost and interconnection matrices in SIGMA algorithm. Therefore, SIGMA permits to perform (within acceptable time, e.g., 1 s) the DMA and DCS structures more complex that ones represented in the tables. The problem is to develop an initial feasible DMA assignment algorithm with polynomial complexity not exceeding the computation complexity of SIGMA.

Cluster algorithm with polynomial complexity described in 5.3.3 can be successfully used for generating an initial DMA assignment. The computation complexity of the cluster algorithm has the same order with SIGMA. We guess, pessimistic approach mentioned above in 5.3.3 and target computer selection strategy with respect to maximum computer resources available will be more successfully for this purpose. Actually, the pessimistic approach is characterized by less probability to obtain a not feasible DMA assignment than the optimistic one. In comparison with the minimum available resources target selection strategy, mentioned above one favors decreasing number of clusters created and, hence, number of appropriate computers used for DMA component allocations, produces better solution for less time. It is illustrated in Tables 5.15 and 5.16

Table 5.15 Strategy to select the next target computer with minimum CPU cycles available

TIGHTNESS	Cluster (optimistic approach)	Time, ms
50	1 200	160
60	1 332	270
70	865	390
80	1 090	457
90	980	540
100	1 400	685

Table 5.16 Strategy to select the next target computer with maximum CPU cycles available

TIGHTNESS	Cluster (optimistic approach)	Time, Ms
50	1 320	260
60	768	270
70	445	390
80	1 340	352
90	980	460
100	1 230	465

6 Performance evaluation and optimization of an adaptive protocol for synchronizing media stream

6.1 Modeling and simulation environment COVERS 3.0

COVERS® is an object-oriented environment aimed to help in the design of distributed and parallel systems. COVERS enables the user to build and simulate the models of such systems. The models are used to evaluate the system correctness and performance, as well as to visualize its behavior.

The main construction unit of a COVERS model is called active object. Active objects are independent concurrently-active event-driven logical machines. COVERS modeling language framework includes diagrams of object structure and interconnection, state-charts as a behavior description, and C++ for data objects and functions.

The structure and behavior of active objects is specified graphically, and the C++ code is added to the specification if required. The graphical constructs are also mapped into C++, so that active object class becomes a normal C++ class. The model is compiled and linked with COVERS runtime C++ libraries to produce the executable. The executable is a normal Windows applications. It can be ported to any computer independently of COVERS.

The user can build model executables of two types: interactive and console. In the interactive model every bit of the graphical specification is animated and is accessible by the user, so that debugging and analysis is done in terms of the original specification. Console models are used for “heavy” simulations where serious statistics are collected.

Compiled active object classes can be grouped into libraries. The basic resource and network models are supplied with COVERS as a standard library set. The user can create his own domain-specific libraries to reduce the modeling process to connecting objects and specifying their parameters.

COVERS is especially suitable to model and analyze systems of various nature and scale that can be characterized as

- Non-terminating
- Continuously interacting with the environment
- Having discrete, event-driven behavior
- Consisting of concurrent interacting components
- Having timeliness an important design issue.

Among others, this domain includes:

- Distributed algorithms and applications
- Client/server systems
- Communication systems
- Computer systems and components
- Networks
- Real-time control systems

The designer of such a system can benefit from using its executable model throughout the whole design cycle in several ways. At the first place, the model will help to overcome the “sequentiality” of human’s way of thinking and to understand what actually happens in the complex concurrent system. Specific correctness issues, such as

- Deadlocks
- Starvation
- Racing
- Critical sections
- Real-time temporal properties can be tested as well as the general functionality. And of course, the designer can investigate the system performance, including
- Response times
- Message latencies
- System throughput
- Resource utilization

In particular, for distributed software applications the executable model will allow to predict the effect of deployment topology on the application performance and find out how good does the application scale.

6.1.1 COVERS 3.0 modeling methodology

The main building block of a COVERS model is called *active object*, or simply *object*. Objects are independent concurrently-active event-driven logical machines. Depending of the level of abstraction, object can represent, for example:

- Piece of hardware
- UNIX process
- Human user
- WAN link
- Client or server
- Physical object

The basic mechanism of object interaction is message passing. (Shared variables and RPC are modeled on top of it.) Whenever an object wants to send or receive a message, it is done through a *port*. The set of ports comprises the object interface.

Object internal structure. We have borrowed some elements of the graphical notation for the object internal structure and the corresponding terminology from ROOM approach [BR96].

Objects may contain other objects to any desired depth. The encapsulated objects run concurrently with each other and with the container object. They can export their ports to the container object interface.

Figure 6.1 shows the internal structure of a simple *Server* object. *Server* contains objects representing two hardware resources, *CPU* and *RAM*, and two software processes which access these resources. Whenever a new transaction arrives at *LocalPort*, the server activates *Main* object. The latter creates a new instance of *Process* object. *Process* executes the transaction (i.e. accesses *CPU* and *RAM* and, probably makes a remote request through a *WANPort*), replies to *Main* and then deletes itself.

This example shows two important features of COVERS model:

Replication of objects. Replicated object represents a collection of objects of the same type connected the same way. A port of a replicated object is equivalent to a collection of ports of all object copies.

Dynamic creation and destruction of objects. Any object can be created and destroyed while the model is running.

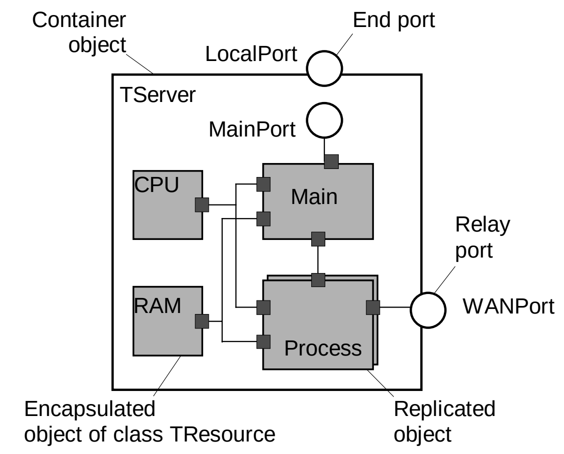


Figure 6.1 Internal structure of a simple server

6.1.2 Object behavior

Object behavior is a part of the object specification that answers the question: when and how does the object react to the external events and conditions. In COVERS the behavior is described in the form of a *sequential statechart.*

Statecharts (an advanced state-machine notation introduced by David Harel's [2]) are accepted as object behavior specification technique by almost all leading OO design methods, including the Unified Method [BKR95]. COVERS supports hyperstates, conditional branching of transitions and history states. To illustrate convenience and expressive power of statecharts, we show COVERS model of the behavior of 100VG station (a part of the Demand Priority protocol, IEEE 802.12 standard), see Figure 6.2.

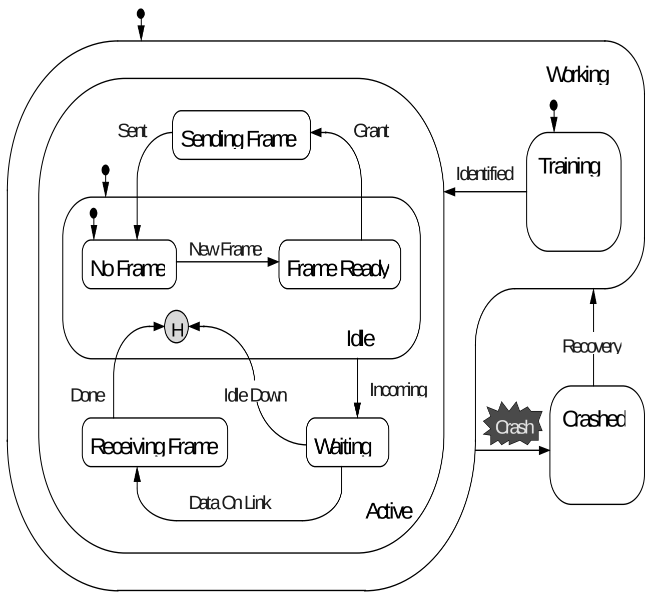


Figure 6.2 Behavior of 100VG station

When the station is switched on, it first executes some *Training* procedure (details not shown) and, having successfully identified itself, comes to an active mode, represented by *Active* hyperstate. While the station is active, it can be either *Idle*, or sending/receiving information. If it is *Idle* and doesn't have any user's frames to send, it is in the *No Frame* state waiting for new frames. When a new frame comes (transition *New Frame*), station goes to the *Frame Ready* state and waits for a *Grant* from the hub. If the *Grant* comes, the frame is transmitted and the station returns to the *No Frame* state. At the same time, no matter whether there is user's frame ready or not, the *Incoming* signal may come (the corresponding transition exits the *Idle* hyperstate and thus works for both *No Frame* and *Frame Ready* states). After the reception is finished, the station returns to where it had been when it was interrupted by the *Incoming* signal. This is modeled by the history state within the *Idle* hyperstate.

At any time during the station operation, it can crash. This is modeled by the *Crash* transition exiting *Working* hyperstate. After *Recovery* it comes back, and, following the initial state marker, starts the training procedure.

Whereas visual appearance of statecharts and the semantics behind hyperstates, etc. is more or less standard and well understood, the exact attributes of states and transitions vary from implementation to implementation. In COVERS each state (primitive or hyperstate) has *entry action* and *exit action*, and each transition has *guard*, *delay* and *action* associated with it. Actions are executed only when a transition is taken in the order shown in Figure 6.3.

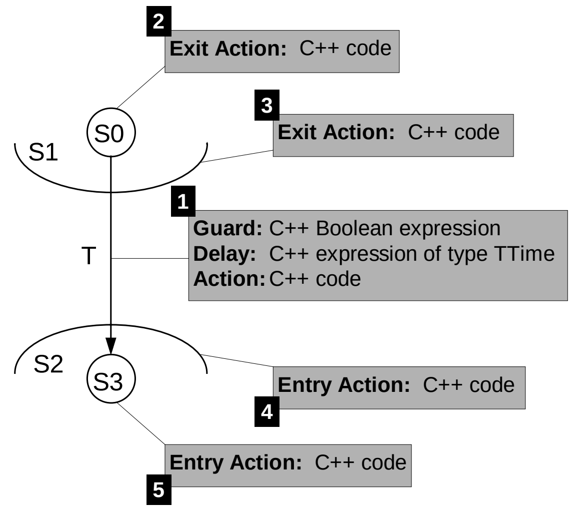


Figure 6.3 Order of action execution

Transition firing takes zero time, hence an object spends all the time in its states. While in a state, an object is passive, nothing is executed. Transition with a delay *t* is taken after it has been enabled (i.e. its guard has been true) for exactly *t*.

Object data members, member functions and data classes used for inter-object communication are specified in C++.

Model semantics. COVERS model has a formally defined semantics based on Timed Transition Systems [Har90] and is fully executable. Timed Transition Systems extend simple and economical interleaving approach to real time. There are two types of steps the model can make: *time steps*, during which the time progresses, but the model state remain the same, and *event steps* when the model state changes instantly, see Figure 6.2. This allows us to handle correctly subtle aspects of concurrent system behavior, such as non-determinism or racing.

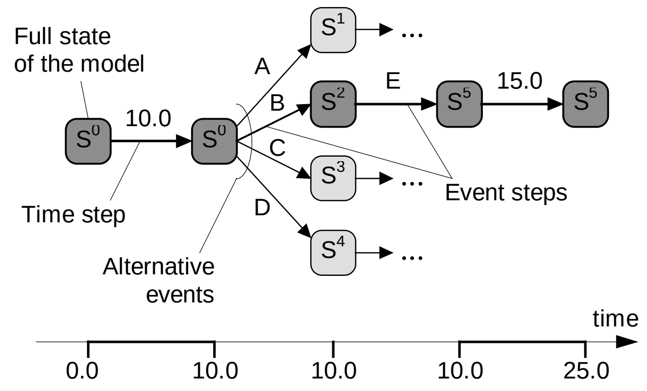


Figure 6.2 Semantical model

In the situations when two or more events are scheduled exactly at the same time (like *A*,*B*,*C* and *D*) most of the simulation tools will always choose the first (in some deterministic order) alternative. COVERS will either make a random choice, or, in the interactive mode, leave it to the user. This ensures that a bigger part of the system state space is covered by a simulation, so it is more likely that an undesirable behaviour is detected.

In general, the formal semantics gives clear answers to the questions like:

- Why does non-determinism appear, and how should it be treated during the model execution?
- Which actions are atomic and which are not?
- What is done instantly and what requires non-zero time?
- What is the semantics of communication of the model components?
- What comprises the global state of the model?

which are important in the system domain we investigate.

6.1.3 COVERS 3.0 environment

After the user inputs the system specification, COVERS generates a C++ code, where object and port types also become C++ classes. The following code, for example, will be generated for *Server* object Figure 1:

```
class TServer: public TObject
  {
  public:
    TServer();

    void Setup();

    TInOutRelay*                 WANPort;
    TInOut< TTransaction >*      MainPort;
    TInOut< TTransaction >*      LocalPort;

    TVectorP< class TProcess > Process;
    class TResource*             RAM;
    class TMain*                 Main;
    class TTimeShareResource*    CPU;
    …
  };

TServer::TServer()
  {
```

```
  Setup();
  }

void TServer::Setup()
  {
  int i, count;
  WANPort   = new TInOutRelay();
  MainPort  = new TInOut< TTransaction >();
  LocalPort = new TInOut< TTransaction >();
  count = ( 10 );
  for ( i = 0; i < count; i++ )
    {
    Process.Add( new TProcess() );
    Process[i]->SetReplicated();
    }
  RAM  = new TResource();
  Main = new TMain();
  CPU  = new TTimeShareResource();
  // Connect ports
  for ( i = 0; i < count; i++ )
    {
    Main->ProcessPort->Connect( Process[i]->Port );
    Process[i]->Port->Connect( Main->ProcessPort );
    }
  RAM->Port->Connect( Main->RAMPort );
  Main->RAMPort->Connect( RAM->Port );

  … // etc.
  }
```

This mapping into C++ makes the modeling technology very flexible, providing, for example, for construction of template object classes, i.e. those parameterized with the classes of other objects.

Before the generated C++ code is compiled, the user is allowed to modify it, so that arbitrary complex model structure or functionality can be achieved.

The complete picture of what COVERS generates and builds is depicted in Figure 6.5.

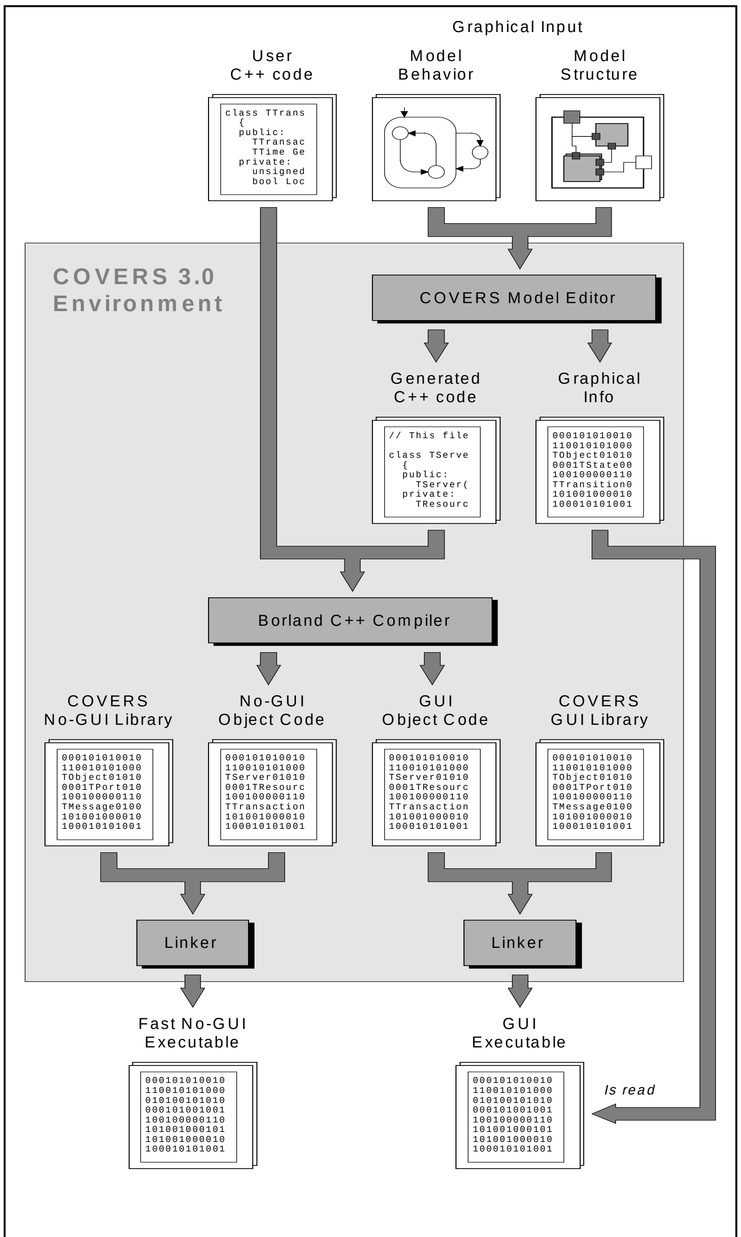


Figure 6.5 COVERS flowchart

6.1.4 Interactive model execution

COVERS enables the user to perform the whole modeling and simulation cycle within a single graphical environment. During the interactive execution every bit of the graphical specification is animated and is accessible by the user, so that model debugging and analysis is done in terms of the original specification. COVERS highlights object interaction, current states and active transitions. Message queues, event lists, active timers are shown and controllable. COVERS collects and displays statistics for each piece of the model, e.g. for sojourn time in a state, queue length in a port, inter-firing interval of a transition.

Traditional debugging tools are also available in COVERS environment, such as

- Inspect

- Breakpoints
- Break conditions
- Logs.

6.1.5 Statistics collection and reporting

COVERS 3.0 C++ class library includes:

Generic parameter class

Classes for statistics collection and reporting for both discrete and continuous statistics

Data set class

Any model parameter can be specified by the user as *iterated* through a set of values. COVERS will perform as many simulation runs as required in order to cover the whole parameter space.

Several kinds of plots can be obtained using COVERS: an observable value versus simulation time, observable value versus iterated model parameter, etc. COVERS can build plots itself or export data sets to other applications, such as MS Excel.

When graphical debugging is no longer needed, COVERS can produce a fast 16 or 32 bit no-GUI executable. This executable can be used for serious statistics collection in long simulation runs.

6.1.6 Comments

We have outlined the modeling approach and a tool with the following features, which set it apart from the existing packages, such as BONeS Designer, SES Workbench, OPNET or Statemate:

Object-oriented. COVERS supports object-oriented modeling methodology and provides for building hierarchical object models of arbitrary complexity and scale.

Statecharts. COVERS supports Statecharts, an advanced notation for object behavior, accepted by all leading OO design methods, including the Unified Method.

C++. COVERS model is completely based on and mapped into C++ - a standard, well-known language.

Formal semantics. COVERS simulation engine is build according to the formally defined semantics and treats correctly subtle aspects of concurrent system behavior, such as nondeterminism, atomicity or racing.

Open. COVERS is 100% open at the level of the C++ code it generates from the user specification, allowing an the user to construct models of arbitrary complex structure and functionality.

Extendible libraries. The library of frequently needed objects is supplied with COVERS and can be easily modified and extended.

Cheap platform. COVERS runs on cheap, highly available platform: 486 or Pentium processor running MS Windows 3.1, Windows 95 or Windows NT.

User-friendly. COVERS enables the user to perform the whole modeling and simulation cycle within a single graphical environment where every bit of the specification is animated and accessible. COVERS offers as much comfort as Borland or Microsoft C++ development environments.

COVERS 3.0 with libraries and examples is available *free* over the Internet for non-commercial use. Watch our site: http://dcn.nord.nw.ru.

6.2 Modeling the Adaptive Synchronization Protocol using COVERS 3.0

The Adaptive Synchronization Protocol (ASP) [RH96] is a typical example of real-time event-based software system. In spite of clear ideas behind ASP its behavior is not fully understandable and it is difficult to predict the optimal policies and the optimal ranges of its parameters for stream synchronization in different environments. So the modeling is the only way to understand the ASP and to analyze its behavior.

6.2.1 A COVERS model of the Adaptive Synchronization Protocol

6.2.1.1 System structure.

The structure of a system where ASP is running is designed using COVERS graphical editor and consists of a set of encapsulated objects: streams, which is controlled by a single controller (see Figure 6.6). The system here is represented as a main active object TSystem, and its encapsulated objects are an object Controller and a replicated object which represents a collection of objects Stream of the same type TStream connected the same way with Controller. A port of a replicated object Stream is equivalent to a collection of ports of all object copies.

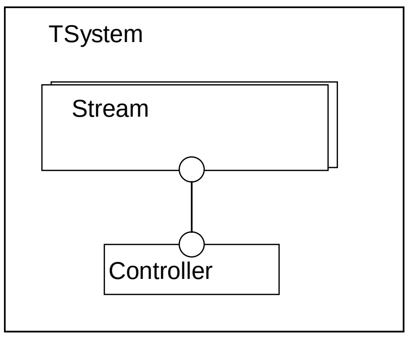


Figure 6.6 ASP structure

Each class TStream encapsulates an object Source (of the type TSource) and a Sink (of the type TSink), communicating through a Channel (of the type TChannel) (see Figure 6.7). Besides, TStream has relay port ControlPort through which Source and Sink communicate with Controller.

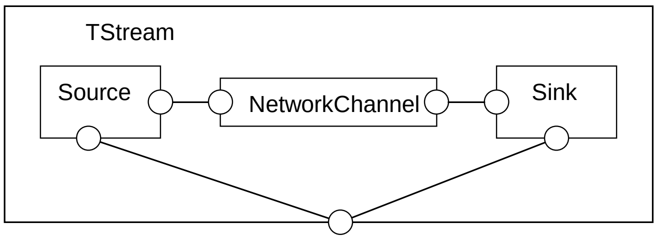


Figure 6.7 Stream structure

Classes TController, TSource, TSink and TNetworkChannel are primitive classes, which don't encapsulate other classes.

6.2.1.2 Class TController

This class has only one input-output port to communicate with Streams. In current version of the model only Start-up protocol and Buffer Control protocol are implemented. TController class behavior is shown in Figure 6.8.

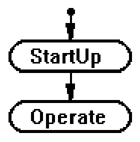


Figure 6.8 Controller behavior

In StartUp state Controller broadcasts Start message contained NominalRate and additional time Delta to all Streams. Both these parameters can be modified at runtime. To do this just right-click on the Controller object in the TSystem Structure window and choose "Modify" entry. Messages which are broadcasted by Controller are of the class TControlMessage, which contains two data objects: NominalRate and Delta.

In Operate state Controller waits for Stream messages to take part in Master/Slave Synchronization protocol and Master Switching protocol.

6.2.1.3 Class TSource

Class TSource is a subclass of TActiveObject class, which is basic COVERS class. Class TSource has two states: StartUp and Generation states (Figure 6.9). In StartUp state Source is waiting for a Control Message, broadcasted by Controller and then comes to a Generation state. While transition, it selects *NominalRate* data from the received message and starts Generation Timer. After activation this timer sends Frame Messages every *1/NominalRate* sec. Generated Frame Messages are stamped by sequential numbers.

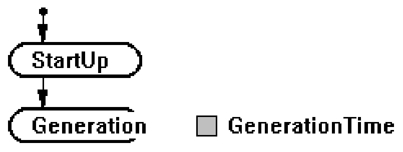


Figure 6.9 Source behavior

6.2.1.4 Class TNetworkChannel

This class models transition delay of incoming messages in network for a random time. Network Channel has the only state and one transition (Figure 6.10). The transition activates when next message comes to it Inbound port. When InMessage transition fires, InMessageAction() function is called. This function creates a dynamic timer, which immediately becomes set. The timer expires after a time which is defined by the exponential distribution with the given *mean* and *deviation*. When the event of expiration happens, the timer calls Expiry() function, which transmits the message to the output Outbound channel port. Both *mean* and *deviation* parameters can be modified at runtime. To do this right-click on the NetwoorkChannel object in the Stream[i] Structure window and choose "Modify" entry.

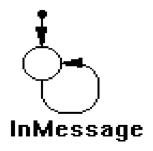


Figure 6.10 Network Channel behavior

6.2.1.5 Class TSink

Input port of TSink class is redefined: its virtual function Receive() defined in such a way, that when a message frame comes to the port, this message is places into a PlayoutBuffer according to its timestamp in increasing order of timestamps.

Class TSink has four states (Figure 6.11). In Initial state it waits for a Control Message, broadcasted by Controller. Receiving a Control message, Sink selects two parameters: *Nominal Rate* and *Delta* which are broadcasted by Controller at the first stage of the protocol.

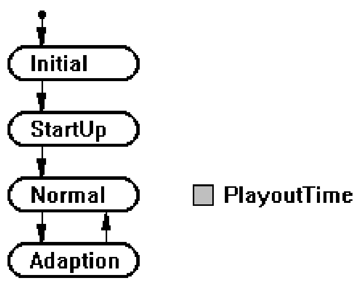


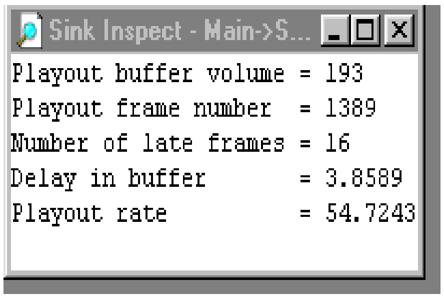
Figure 6.11 Sink behavior

Delta defines a delay before playout start. This value stands in the *delay* entry of a transition from StartUp state to Normal state. When this transition fires, Playout Timer is activated with the delay *1/PlayoutRate*. When the Timer expires, the next frame is released from the PlayoutBuffer. Playout Timer then periodically releases next frame from the buffer with the period *1/PlayoutRate.* PlayoutRate is normally equal to *NominalRate* but in Adaptation phase is changed according to the protocol.

6.2.1.6 Visualisation

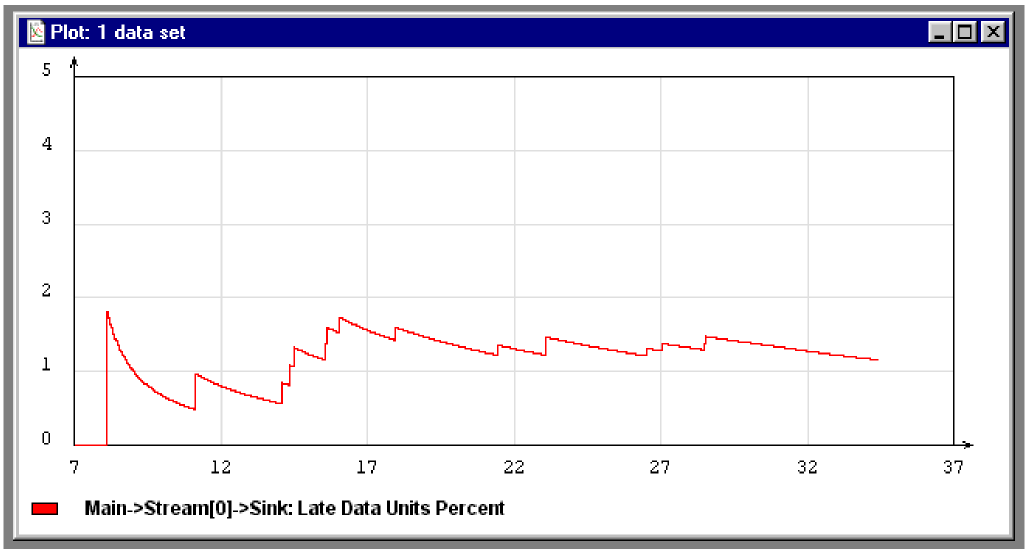
COVERS has various facilities to collect and display data during the simulation. In Inspect window the following data are shown during the simulation:

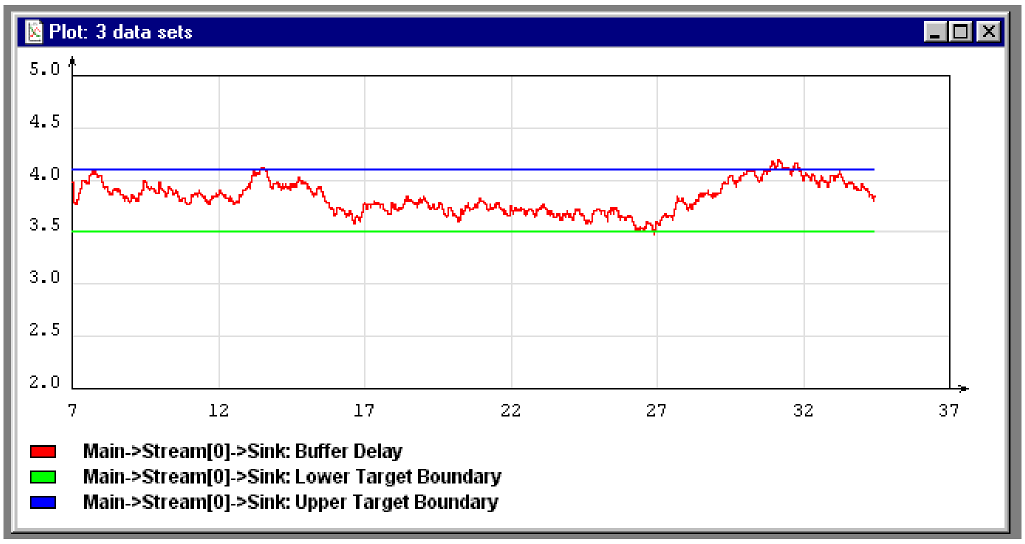
- Playout buffer volume;
- Playout frame number;
- Number of late frames;
- Delay in buffer;
- Playout rate.

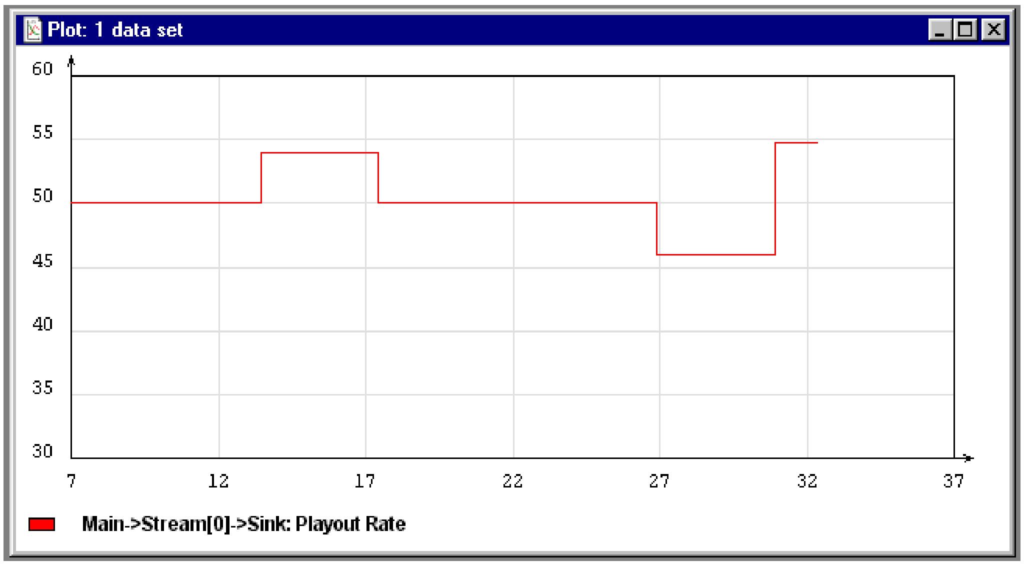


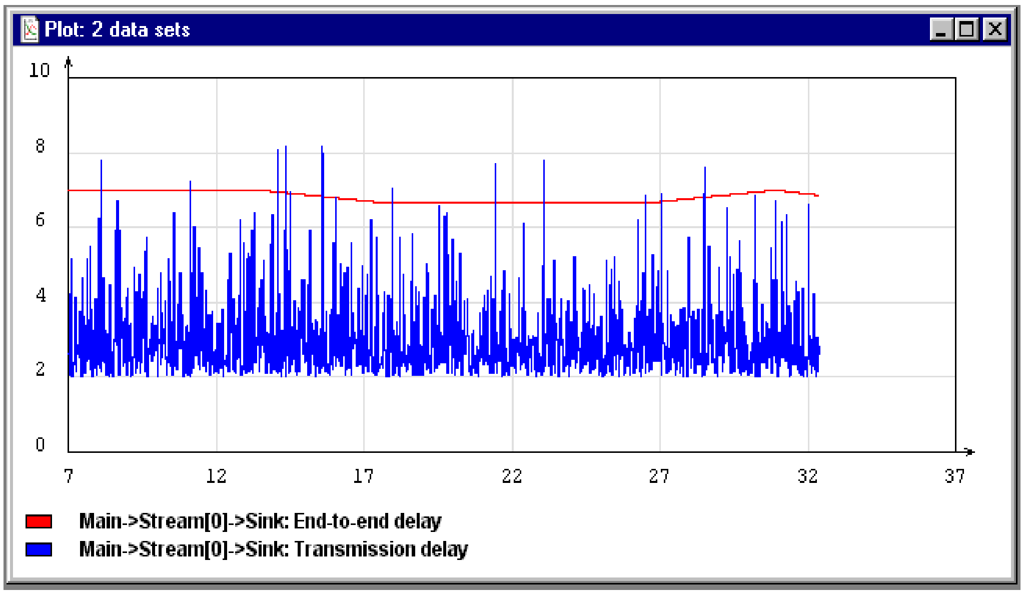
```
Playout buffer volume = 193
Playout frame number  = 1389
Number of late frames = 16
Delay in buffer       = 3.8589
Playout rate          = 54.7243
```

Besides, for the analysis user can define plots to show time dependencies of every parameter. Some plots, which were used during analysis of the ASP, are shown below.









The user can define own data sets and watch them during simulation run.

6.2.2 Future work

Analysis of the two first phases of Adaptive Synchronization Protocol has shown that the COVERS model is really convenient for understanding of the ASP behavior and examining its characteristics. The following problems has to be solved at the next stage of research:

1. Examining two first stages of the ASP for real multimedia streams and real distribution of transmission delay.
2. Elaborating of optimization criteria for synchronization of media streams, which includes: End-to-end delay, percent of late data units in a stream, jitter and skew.
3. Design of optimization procedure, which finds optimal parameters for synchronization of multimedia streams on the base of the ASP.
4. Investigation of multicast protocols (ABCAST, CBCAST and GBCAST) for correct implementation of Master Switching Protocol.

7 Platform for configuration and synchronization management investigation

We propose to use prototype of videoconferencing system as platform for configuration and synchronization investigation.

Videoconferencing systems like many distributed multimedia applications (video-on-demand, multimedia mail and so on) are known for high requirements they put on computational power of DCS computers and bandwidth of communicational resources in a DCS and therefore can be considered as a typical resource-intensive distributed applications. That's why to organize videoconferencing means to apply the newest multimedia technologies in the field of signal processing and recognition, visual communications, integrated digital networking and so on. And that is why a software system based on a videoconferencing technology may be used as a robust sample platform to test various multimedia solutions, techniques and protocols.

Abstract software to support videoconferencing should meet the following requirements:

- capable to set up videoconferencing connections
- can handle hardware and software collisions
- has means to synchronise multimedia streams on the client's end
- capable to manage communicational resources in a DCS
- capable to reconfigure a videoconferencing structure

The existing videoconferencing systems (Intel ProShare(™) Personal Conferencing, LANDesk™ Personal Conferencing Manager, Lotus RealNotes etc.) solve most of the above-mentioned problems. However, they are pretty hard to manage on a system level and too enormous and universal to use as an easily manageable and configurable test platform for distributed applications and protocols.

We propose a prototype of the simplest distributed videoconferencing application with the following characteristics:

- client-server architecture based and implemented on so-called mailslots mechanism
- arbitrary number of "dynamic" clients ("dynamic" clients connect to and disconnect from a server any time they want)
- arbitrary structure of multimedia streams (video/audio/sequence of pictures/text/etc.) broadcasted from a server to all connected clients
- 100 % open at C++ level

Mailslots mechanism belongs to Win32 API and allows to:

- create a mailslot with a given name – an entity to receive the data - on a client's end of a system
- send the data packets of arbitrary structure from a server's end of a system
- broadcast the data to all mailslot with the same name
- regulate the size of a buffer on both ends of a stream

At present we plan to use this prototype as a platform to organize the simplest video-education system where a server-teacher broadcasts two concurrent streams – a stream of pictures and a stream of comments to those pictures – to all connected clients-pupils. Adaptive synchronization protocol (see

6) will be embedded in the prototype to synchronize the two above-mentioned streams on every client computer.

We plan to manage:

- a number of active clients-pupils
- a bandwidth for every channel “teacher-pupil”
- a size of a buffer for every client computer
- ASP parameters (low/high water mark for every client computer, target zone for master stream “teacher-pupil” and so on)

We plan to measure:

- end-to-end delay for every client stream
- jitter for every client stream
- a skew between two concurrent stream for every client-pupil
- percentage of pictures and comments lost for every client
- number of service ASP messages sent over the network by clients

In the near future we plan to use this prototype as a platform to model mapping algorithms with simultaneous instantiation of the DMA components (clients) and visual control of QoS degradation/improvement.

8 Conclusions

At present, audio, video, graphics, image processing and real-time processing are not new areas. However, as argued in [StNa95], the simple composition of existing systems and methods is not a global multimedia solution. Particularly, the management problems of configuring DMA and synchronizing media streams, the subject of the project OPTIMUS, need to be developed and are such the examples of research and development directions which will dominate today and in the nearest future.

The main research and practical results obtained within the OPTIMUS project are as follows:

1. System-approach-based methodology for distributed multimedia application management is proposed. A management architecture supported by CINEMA system includes configuration, session and synchronization management subsystems described.

2. Management tasks that have to be solved during the preparation, establishment, active and termination phases of distributed application lifecycle are considered and analyzed. It was shown, that the management system necessarily has to support solving some management tasks, such as QoS (re)-negotiation, application-to-computer system (re)-mapping more than once and at different phases of the application lifecycle

3. QoS architecture provided by CINEMA development platform is described. Application clients can specify QoS request as QoS ranges in order to allow the best possible QoS choise by configuration service. Application level negotiation has to encompass all components and links of DMA flow graph model, even if resource reservation is not performed. Application level negotiation differs from transport level negotiation. It requires a separation of QoS abstraction levels for media specific and transport level characteristics and definition of mappings between them.

4. Configuration service requirements, functions and algorithms are considered in details. Requirements to the configuration service include following supports:
   - for media specific QoS parameters,
   - for port type compatibility for DMA logical topology,
   - for various QoS due to limited functional capabilities of components, computers and resource availability of the computers,
   - for media parameter format constraints associated with each component port. It is important to note that the format constraints relate both to components and computers, and they have to be specified for each (component, computer) pair,
   - for stream relations,
   - for complex DMA topologies.

5. Configuration service architecture proposed covers the first two phases of the DMA lifecycle, i.e. preparation and session set-up phases, and consists of three subsystems:

   - application handler that is located on the system the client is established and services the client to build up a DMA flow graph at the preparation phase,

   - configuration server as centralized entity that defines DMA physical topology, i.e. it negotiate media values in the DMA flow graph for given QoS requirements, maps the DMA flow graph to the DCS and instantiates the DMA components in the DCS computers chosen,

   - configuration handler as an entity distributed over DCS, based on extended negotiation and resource reservation protocol XNRP and performing the final QoS negotiation, completes the session set-up phase with resource reservation for components and links of the DMA topology.

6. The general configuration service algorithm describing functions and interconnections of all three above-mentioned parts above is presented. The algorithm for configuration server and its input and output data are refined.

7. Problem of mapping a DMA to a DCS is described and its mathematical formulation is proposed. Constructive and iterative algorithms that differ in computation complexity and relative error are developed, described and implemented in C++. All algorithms proposed can perform mapping for:
   - the arbitrary topologies of distributed multimedia applications,
   - the arbitrary topologies of distributed computer systems,
   - computation and communication resource constraints of the DCS,
   - multicasting,
   - allocation dependency of computational resource requirements of adjacent DMA components mapped to the same or to different computers.

8. The technology for experimental analysis of efficiency of the mapping algorithms is proposed. The technology is based on a set of testable graph structures each of that can be parameterized randomly and replicated. The structures of graphs proposed are typical for real DMA flow graphs. Program generators for these types of DMA graphs are implemented in C++.

9. Experimental analysis of computation complexity and solution error of the mapping algorithms developed was performed using the generators of random DMA flow graphs. Experiments results were presented in tables and discussed. The exact searching algorithm based on the branch-bound method with exponential complexity was used to obtain the exact solution. The heuristic algorithm based on cluster analysis with polynomial complexity is useful to obtain an acceptable DMA assignment. The iterative algorithm SIGMA that has polynomial complexity allows improving an initial feasible DMA placement. For all experiments performed on Pentium 166 MHz 32 MB RAM, Windows 95 the total computational time of cluster and SIGMA algorithms was less than 1 s with solution error less than 6.3%.

10. Modeling methodology based on the object-oriented environment COVERS® developed by the research team of the Technical University of St. Petersburg is described. COVERS is aimed to help an user to model and design distributed and parallel systems. It enables the user to build and simulate the models of these systems. The models are used to evaluate the system correctness and performance, as well as to visualize its behavior. The methodology and COVERS tool were used for model design and modeling of the Adaptive Synchronization Protocol for distributed multimedia systems. (The protocol was developed by the research team of the University of Stuttgart).

11. Analysis of the two first phases of Adaptive Synchronization Protocol has shown that the COVERS model is really convenient for understanding of the ASP behavior and examining its characteristics. The following problems has to be solved at the next stage of the research:

    - Examining two first stages of the ASP for real multimedia streams and real distribution of transmission delay.

    - Elaborating of optimization criteria for media streams synchronization, which includes: end-to-end delay, percent of late data units in a stream, jitter and skew.

    - Design of optimization procedure, which finds optimal parameters for synchronization of multimedia streams on the base of the ASP.

    - Investigation of multicast protocols (ABCAST, CBCAST and GBCAST) for correct implementation of Master Switching Protocol.

12. Testbed platform for the configuration and synchronization management mechanisms of distributed multimedia applications investigation is described, and its functions and aims are discussed. A prototype of the distributed videoconferencing application with the following characteristics is proposed:
    - client-server architecture based and implemented on so-called mailslots mechanism

- arbitrary number of “dynamic” clients (“dynamic” clients connect to and disconnect from a server any time they want)
- arbitrary structure of multimedia streams (video/audio/sequence of pictures/text/etc.) broadcasted from a server to all connected clients
- 100 % open at C++ level

In the future we would like

- to refine the mapping algorithms,
- to continue the mapping algorithms efficiency analysis on the base of extended set of real DMA topologies,
- to combine the mapping algorithms with negotiation and resource reservation protocol such that to cover the main phases of the DMA configuration management,
- to model and to optimize the adaptive synchronization protocol,
- to implement the DMA testbed platform for the configuration and synchronization management mechanisms investigation.

9 References

[AH+97] M.Alexandrov, A.Hagin, N.Hagin, Yu.Karpov, V.Voinov. Quality of Service Management Architecture for Distributed Applications. *Proc. International Conf. on Informatics and Controls ICI&C'97*, St. Petersburg, Russia, 1997.

[AH+97a] M.Alexandrov, A.Hagin, N.Hagin, Yu.Karpov, V.Voinov Quality of Service Management of the World Wide Web. *Proc. 4th Workshop of the Openview University Associstion OVUA'97*, Technical University of Madrid, April 2 -4, 1997.

[Bar96] I.Barth. Configuring Distributed Multimedia Applications Using CINEMA. *IEEE Workshop on Multimedia Software Development MMDS 96,* Berlin, Germany, 1996.

[BCA+92] G.Blair et al. An Integrated Platform and Computational Model for Open Distributed Multimedia Applications. *In 3rd International Workshop on Network and Operating System Support for Digital Audio and Video,* p. 209-222, 11, 1992.

[BKR95] A.V.Borshchev, Yu.G.Karpov, V.V.Roudakov. *COVERS - A Tool for the Design of Real-Time Concurrent Systems*. In: V. Malyshkin (Ed.). Parallel Computing Technologies. Proceedings of the 3rd International Conference PACT-95, Lecture Notes in Computer Science No 964, Springer, 1995.

[Bok87] S.H.Bokhari. Assignment Problems in Paralel and Distributed Computing. Kluwer Academic Publ., 1987

[BR96] G. Booch, J. Rumbaugh. *Unified Method for Object-Oriented Development.* Documentation Set Version 0.8. Rational Software Corporation, 1996.

[DFBR95] G.Dermler, W.Fiederer, I.Barth, K.Rothermel. *A Negotiation and Resource Reservation Protocol (NRP) for Distributed Multimedia Systems.* Fakultätsbericht 11/1995, Technical Report, University of Stuttgart, (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html

[DFR95] G.Dermler, W.Fiederer, K.Rothermel. *A Framework for Negotiable Quality of Service in Distributed Multimedia Systems*. Fakultätsbericht 10/1995, Technical Report, University of Stuttgart, (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html

[DKPS95] M.Degermark, T.Köhler, S.Pink, O.Schelen. Advance Reservations for Predictive Service. – *Proc. of 6th International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, NH, USA, April 1995

[FGV95] D.Ferrari, A.Gupta, G.Ventre. *Distributed advance reservation of real-time connections*. – Technical report TR-95-008, Tenet Group, University of California at Berkeley, and International Computer Science Institute, 1995 (http://www.icsi.berkeley.edu)

[Har87] D. Harel. *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, Vol. 8, No. 3, June 1987.

[Har90] D. Harel et al. *STATEMATE*: A Working Environment for the Development of Complex Reactive Systems*.* *IEEE Transactions on Software Engineering* , Vol. 16, No. 4, April 1990.

[HDR96a] A.Hagin., G.Dermler, K.Rothermel. *Mapping of Distributed Multimedia Applications Based on a Sequential Method*. *Systems.* Fakultätsbericht 16/1996, Technical Report, University of Stuttgart, (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html)

[HMP92] T. Henzinger, Z. Manna, A. Pnueli. *Timed Transition Systems.* Technical Report TR 92-1263, Dept. of Computer Science, Cornell University, January 1992.

[HRD96] A.Hagin., G.Dermler, K.Rothermel. *Problem Formulations, Models and Algorithms for Mapping Distributed Multimedia Applications to Distributed Computer Systems.* Fakultätsbericht 3/1996, Technical Report, University of Stuttgart, (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html)

[IH96] M.A.Iqbal, A.Hagin. *Partitioning and Mapping Techniques for Distributed Multimedia Applications.* Fakultätsbericht 14/1996, Technical Report, University of Stuttgart, (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html)

[IMA93] HP Company and IBM Corporation and SunSoft Inc. Multimedia System Services, Version 1.0, available via ftp from ibminet.awdpa.ibm.com, 7 1993.

[KH+95] Käppner et al. Eine verteilte Entwicklungs- und Laufzeitumgebung für multimediale Anwendungen. In *Proceedings of KiVS'95,* p. 76-86, 1995.

[Min86] M.Minoux. Mathematical Programming. Theory and Algorithms. John Wiley and Sons, 1986

[NaSm95] K.Nahrstedt, J.Smith The QoS Brocker. *IEEE Multimedia*, Vol. 2, No. 1, Spring 1995.

[RBF96] K.Rothermel, G.Dermler, W.Fiederer. A communication Infrastructure for Multimedia Applications. *European Conference on Networks and Optical Communications,* Heidelberg, Germany, 1996.

[RBH94] K.Rothermel, I.Barth, T.Helbig. *Architecture and Protocols for High-Speed Networks,* Chapter CINEMA – An Architecture for Distributed Multimedia Applications, p.p. 253-271. Kluwer Academic Publishers, 1994.

[RDF97] K.Rothermel, G.Dermler, W.Fiederer. *QoS Negotiation and Resource Reservation for Distributed Multimedia Applications.* Technical Report, (http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html)

[RH96] K.Rothermel, T.Helbig. An Adaptive Protocol for Synchronizing Media Streams. *ACM/Springer Multimedia Systems*, 1996.

[RH96a] K.Rothermel, T.Helbig. Clock Hierarchies: An Abstraction for Grouping and Controling Media Streams. *IEEE Journal on Selected Areas in Communications – Synchronization Issues in Multimedia Communications,* 1996.

[RPP94] *Routing, Placement, and Partitioning,* edited by George W.Zobrist. Ablex Publ. Corp., 1994

[SGW94] B. Selic, G. Gullekson, P.T. Ward. *Real Time Object-Oriented Modeling.* John Wiley & Sons, Inc. 1994.

[Sol91] G.V.Soldatenko. *Sequential method for solving extreme combinatorial problems.* Novosibirsk: Science, 1991, (in Russian)

[StNa95] R.Steinmetz, K.Nahrstedt. *Multimedia: Computing, Communications and Applications*. Prentice Hall PTR, 1995.

[WDS^{+}95] L.C.Wolf et al. Issues of Reserving Resources in Advance. *Proc. of 6th International Workshop on Network and Operating System Support for Digital Audio and Video,* Durham, NH, USA, April 1995.

10 Appendix 1. Mapping problem specification language

We propose the following data format to specify a DMA graph including components and links as well as a DCS graph formed by computers and virtual channels routed over one or more communication resources. In a DMA graph nodes represent components that are interconnected by arcs representing links (i.e. data streams) between components. The DCS graph shows possible virtual channel connections between the computers of the DCS. The mapping problem itself is described by means of a simple presentation language. Proposed language has a limited set of operators each of them allows to declare the certain DMA or DCS entities such as the DCS computers or the DMA components.

10.1 DMA graph presentation

10.1.1 DMA component specification structure

The DMA component is represented by a simple record containing the information about computational resources required by this component (i.e. CPU cycles per second needed by this component to run normally) and the amount of free memory needful to launch this component. In so-called heterogeneous situation one has to specified the separate requirements for each DCS node this component can be assigned to.

The common syntax of such a structure looks as follows:

- in homogenous situation:

  component component_name(cpu_req, mem_req);

- in heterogeneous situation:

  component component_name(computer, cpu_req, mem_req);

The examples of various component specifications are given below:

- for homogenous situation:

  component mixer(120, 50);

This operator declares the component named “mixer” that requires 120 units of computational resources and 50 memory units.

- for heterogeneous situation:

  component mixer(P100, 120, 50);

  component mixer(P120, 100, 40);

This operator declares the component named “mixer” that requires 120 units of computational resources and 50 memory units if assigned to the DCS computer named P100 and 100 CPU units with 40 memory units if assigned to P120.

10.1.2 DMA link specification structure

The DMA link is represented by a record as well. Firstly, such a record holds the information about the components (a source and a sink) this link is formed by. Secondly, the bandwidth required by this link is specified. And finally, one have to specify CPU and memory units needed by the source and the sink components to transfer (i.e. to send and to receive) data through the link.

The syntax of such a structure looks as follows:

link (source_name, sink_name, src_cpu_req, src_mem_req, rcv_cpu_req, rcv_mem_req, bandwidth);

And below follows the example of a link structure:

link (source, sink, 10, 5, 5, 2, 5);

This operator declares a link with the source component named "source" that requires 10 CPU units and 5 memory units to send the data and the sink component named "sink" that requires 5 CPU units and 2 memory units to receive the data. Communication requirements of this link are set to 5 data units per second.

10.2 The DCS graph presentation

10.2.1 DCS computer specification structure

An abstract DCS computer is supposed to have certain types of available resources. They are: computational resources measured in CPU cycles per second this computer can perform, memory units available on this computer and communication resources measured in data units per second. The costfactors (i.e. the financial cost of a resource unit) for each type of the above-mentioned resources have to be specified as well.

The syntax of such a structure is specified as:

computer computer_name(cpu_cap, cpu_cost, mem_cap, mem_cost, io_cap, io_cost);

The example computer structure looks as follows:

computer P100 (100, 1, 64, 1, 50, 2);

Such an operator declares the DCS computer named "P100" that has 100 CPU units, 64 memory units and 50 "communication" units available. The costfactors for this computer's resources are 1, 1 and 2 correspondingly.

10.2.2 DCS virtual connection specification structure

A virtual channel connection (VC) is a directed logical connection between two DCS computers with assigned communication capacity. Actually, a VC is routed over one or more physical links of the DCS (network segments, networks) and the available capacity of a VC is equal to the minimum available capacities of all DCS communication resources this VC is routed over. Proceeding from the above considerations we can propose the following VC presentation structure:

vc (computer1, computer2, commrc1, commrc2 … commrcN);

This structure may be illustrated by the following example:

vc (P100, P120, LAN1, LAN2);

This operator declares a virtual channel connection between the computers "P100" and "P20" routed over the networks: LAN1 and LAN2. Note that communication resource (e.g., LAN1 or LAN2) may be shared by one or more else virtual connection and its capacity may be distributed among all VCs traversing this resource.

10.2.3 DCS communication resource specification structure

Communication resource is a physical link in a DCS. It may be specified by communication capacity measured in data units per second that may be transferred through this resource and the costfactor (i.e.

the financial cost of each data unit transferred by means of this resource). The possible communication resource presentation follows:

commrc commrc_name (bandwidth, cost);

An example of the commrc structure follows:

```
commrc LAN ( 100, 1 );
```

We have just declared a commrc named “LAN” with communication capacity equal to 100 and the costfactor equal to 1.

10.3 Initial assignment specification

The INITIAL assignment presentation format is required to specify a **feasible** initial mapping variant of a DMA graph to a DCS. Such a language is needful only for a certain class of mapping algorithms that are designed to improve the given assignment (e.g. the algorithm proposed in 1.4).

The common syntax of the “assignment” operator looks the following way:

assign component_name to computer_name;

An example of this operator is shown below:

```
assign mixer to P100;
```

This operator assigns the component named mixer to the DCS computer named P100.

10.4 Example of specification file

```
[DCS]

computer A ( 7, 1, 7, 1, 7, 1 );
computer B ( 7, 2, 5, 2, 6, 2 );
computer C ( 7, 3, 4, 1, 8, 0 );
computer D ( 7, 2, 10, 1, 10, 3 );

commrc AB ( 6, 2 );
commrc AC ( 6, 3 );
commrc AD ( 6, 1 );
commrc BC ( 6, 1 );
commrc BD ( 6, 4 );
commrc CD ( 6, 2 );

vc ( A, B, AB );
vc ( A, C, AC );
vc ( A, D, AD );
vc ( B, C, BC );
vc ( B, D, BD );
vc ( C, D, CD );

[DMA]

component a (A, 2, 1);
component a (B ,2, 0);
component b (3, 1);
component c (4, 1);
component d (2, 1);

link ( a, c, 1, 1, 1, 1, 2 );
link ( b, c, 3, 2, 1, 2, 1 );
link ( c, d, 2, 2, 1, 2, 1 );

[INITIAL]

assign a to A
assign b to B
assign c to C
assign d to D
```