3
Design and Optimization of High-Performance Protocols with the DO-IT Toolbox

Andreas Mitschele-Thiel
Friedrich-Alexander-Universität Erlangen-Nürnberg,
Lehrstuhl für Informatik VII, Martensstraße 3, 91058 Erlangen, Germany,
email: mitsch@informatik.uni-erlangen.de

Peter Langendörfer
Brandenburgische Technische Universität Cottbus, Institut für Informatik,
Lehrstuhl für Rechnernetze und Kommunikationssysteme, Postfach 10 13 44,
03013 Cottbus, Germany, email: pl@informatik.tu-cottbus.de

Ralf Henke
Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, Institut für Rechnernetze und Betriebssysteme, Postfach 4120, 39120 Magdeburg, Germany, email: henke@cs.uni-magdeburg.de

Abstract

In the telecommunication industry, the *Specification and Description Language* (SDL) is a widely accepted technique to support the software development process. While several commercial SDL tools exist that focus on functional aspects, rather little research has been done concerning the integration of nonfunctional aspects in the development process.

Our research is focusing on the integration of performance aspects in the development process. In the paper, we give an overview on the DO-IT toolbox and describe how the toolbox can be applied to develop a parallel implementation of a multimedia application on top of the XTP protocol suite. The DO-IT toolbox supports the formal specification of performance requirements (e.g. response time and throughput) and the selection of the appropriate design and implementation decisions.

Keywords

FDT-based system and protocol engineering, FDT-based implementation, parallel systems, SDL, MSC, performance modeling and optimization, tool support, XTP, multimedia

1 INTRODUCTION

SDL'92 (ITU, 1993), with it's support for object orientation, supports the software engineering process from object-oriented design down to the generation of executable code. In conjunction with Message Sequence Charts (MSCs) (ITU, 1993a), system simulation and testing are supported, too. Besides a number of proprietary tools and tools from academia, there are two main providers of commercial tools for SDL, namely Telelogic with SDT (Telelogic, 1995) and Verilog with GEODE (Verilog, 1994). The tools support formal specification, validation, simulation, code generation and testing. While the tools for specification, validation, simulation and testing are widely used, the generation of the implementation is often done manually. This is due to the inefficiency of the code generated by the tools. In addition, implementations generated by the tools typically consume considerably more memory. In contrary, the manual implementation of SDL specifications contradicts the intended purpose of SDL and forces intensive testing of the application at the implementation level in order to ensure consistency with the specification.

A related problem is the lack of a formal approach in the system development cycle that supports non-functional requirements, e.g. performance or fault-tolerance requirements. This becomes even more obvious when an SDL specification is implemented on parallel systems due to the wide variety of design decisions that have to be met. These design decisions include (but are not limited to) the architecture of the parallel system, the distribution of code and data as well as the strategies employed for scheduling and dynamic load balancing.

In the paper, a set of tools supporting our methodology is presented. Our approach is focusing on the development of high-performance parallel systems with SDL and MSCs. The approach fully integrates performance issues in the system development cycle. We show the applicability of our approach by applying the tools to the implementation of a multimedia conference application based on the Xpress Transfer Protocol (XTP) (Strayer et al, 1992).

The topic is highly relevant since it allows for the rapid development, configuration and modification of parallel systems in the scope of SDL, which provide the required performance. Especially in telecommunications, a highly competitive market, the time to market has become the major issue to ensure competitiveness. The use of parallel systems is a must to provide the performance required by multimedia applications, especially at the server side.

The paper is organized as follows. In section 2, the DO-IT toolbox supporting the integration of performance aspects in the development process is described. In section 3, the application of the DO-IT toolbox to design a parallel multimedia application is described. Conclusions are given in section 4.

2 THE DO-IT TOOLBOX

2.1 Outline of the Underlying Methodology

The major goal of our approach is the early and systematic integration of performance aspects into the development process. This allows for minimizing time and cost for redesign and reimplementation. The outline of the methodology is depicted in figure 1. Note that the development methodology is not limited to the waterfall approach. Rather, the feedback arcs are omitted for simplicity of the figure.

Starting point of the methodology is the requirements specification. The formal part of the requirements specification comprises two aspects, the functional and non-functional requirements. Our approach is based on use cases. Use cases are specified in the MSC notation. The functional requirements are formally specified with standard MSCs. The performance requirements of the system under development are also given formally. For this, an annotated extension of MSCs is used in conjunction with a high-level notation to formally specify the interrelation between a set of MSCs in respect to performance aspects. In addition, the machine architecture is formally

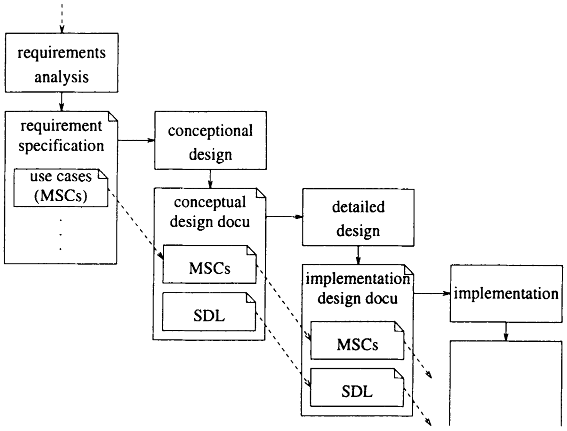


Figure 1 Outline of the development methodology

specified with respect to structural, behavioral and performance aspects. The formal specification of functional and performance aspects as well as the machine architecture are a prerequisite for the automization of the design process.

The conceptual design specification is derived from the requirements specification. The structural (conceptual) design is specified with SDL. The implementation design is derived in a series of steps subsequently moving from a purely structural to a detailed behavioral design document. In conjunction with the refinement of the SDL specification, the MSCs are subsequently refined to reflect the internal behavior of the refined SDL specification.

A more detailed description of our methodology with its support for performance analysis and system synthesis can be found in (Mitschele-Thiel, 1996).

2.2 Overview on the DO-IT Toolbox

In order to support our methodology, especially the integration of performance aspects in the design and implementation process for parallel systems, the DO-IT toolbox (Design and Optimization – Integrated Toolbox) has been devised. As depicted in figure 2, the DO-IT toolbox consists of three major components, namely MODE (MOdel DErivation), MOPS (Model based Optimization of Parallel Systems) and COPS (Code Optimization for Parallel Systems). The three components of the DO-IT toolbox are intended to complement the commercially available tools for SDL and MSCs.

The input to the MODE component consists of the SDL specification and the *load model.* At this stage of the development cycle, the load model formally specifies the performance requirements of the system under development. As already mentioned, we employ use cases to formally specify the performance requirements.

The use cases are specified in the MSC notation. An additional notation has been introduced to specify the interrelationship between the MSCs. Since we employ a notion similar to high-level MSCs (HMSC) currently in the standardization process, we call our notation high-level performance MSC, or HPMSC for short. A HPMSC specifies all the performance requirements of a system under development in a similar way a HMSC specifies the functional requirements of a system. A HPMSC is structured as a tree. The leaves of the tree refer to MSCs in most cases.

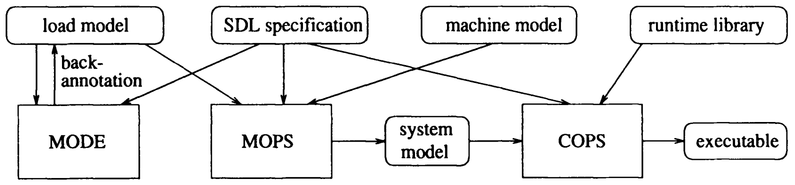


Figure 2 The DO-IT toolbox

A detailed description of the HPMSC notation is given in section 3.2. An example of an HPMSC to specify the performance aspects of our multimedia application is given in figure 6.

Given the load model, i.e. the formal specification of the performance-relevant use cases, and the corresponding SDL specification, the MODE component is employed to derive the *performance data* of the SDL specification. Performance data denote the cost of executing the given use cases on the target system. In addition to the derivation of the performance data, MODE is responsible for the back annotation of the load model with the respective performance data. Thus, the performance data are used to annotate the MSCs referred to in the HPMSC. As described in detail below, two different instances of MODE exist, depending on the level of detail provided by the given SDL specification. Thus, MODE may be applied in early as well as in late design phases.

Once the necessary performance data have been derived, the MOPS component can be applied. The major inputs to MOPS consist of the load model and the machine model. After the execution of MODE, the load model contains all required performance information of the application. MOPS computes the major design decisions concerning the software, the hardware as well as the mapping of the software on the hardware. The resulting design decisions are given in an abstract form. This is denoted as system model.

The system model represents the basis for the code generator and optimizer COPS which is responsible for the implementation of the design decisions made by MOPS. In the paper, we focus on the components MODE and MOPS.

2.3 Derivation of Performance Data – MODE

The derivation of performance data for a given SDL specification, especially the execution and communication cost of its implementation is supported by the MODE tools, namely MODE-M and MODE-A. The two tools support the derivation of performance data at different phases in the development cycle. Both tools support the back annotation of the MSCs with performance data. With MODE-M (MOdel DErivation by Measurements), the performance data are derived by measuring the computation cost of the SDL specification on a specific machine. In this context, the term "machine" denotes the combination of hardware and system software that executes the SDL constructs. Thus, the computation and communication cost with which the use cases (MSCs) are annotated reflect the cost of execution of the constructs with the given system software on a specific hardware unit. Note that the cost also depend on the code generator and compiler that are used. Thus, each combination of code generator, compiler, system software and hardware constitutes a separate machine. In case several machines are at the disposal of the design process, the MSCs are annotated with vectors where each element of the vector reflects the cost of the corresponding SDL constructs on a specific machine.

The derivation of the performance data with MODE-M is performed in a series of steps, itself involving a set of tools. First, the given SDL specification is automatically instrumented, i.e. additional instructions are integrated into the code that record start and end time of the relevant SDL constructs. The instrumentation is controlled by the corresponding MSCs. Our current instrumentation tool performs the automatic instrumentation of C code (Dauphin, Dulz and Lemmen, 1995). However, with support of the code generator for SDL, e.g. as provided by the SDT code generator, the respective instructions can be directly integrated into the SDL specification. This eliminates the need to associate each SDL construct with the respective parts in the C code.

After instrumentation, the code is translated and executed on the target hardware. During execution, the performance-relevant data are traced. The execution of the code is controlled by the input signals corresponding to the respective MSCs. As a result, only those parts of the SDL specification are typically executed for which a respective MSC exists. Thus, no performance data are traced for the remaining parts of the SDL specification. However, this is not a problem as long as the performance-relevant parts of the SDL specifications are covered by MSCs, which is typically the case.

The monitoring of the system and the recording of the traces is either done in software or with the ZM4 hybrid monitoring system (Dauphin et al, 1994). The ZM4 allows for the monitoring of parallel as well as distributed systems and supports a wide range of hardware interfaces. For the analysis of the traces, the SIMPLE analysis tools (Dauphin et al, 1994) are used. An additional tool is needed that supports the back annotation of the respective MSCs with the measured performance data. As an example of the annotation of an MSC with performance data, the annotated version of the MSC depicted in figure 8 is given in figure 11.

Instead of the derivation of the performance data by means of measurements, the analytical modeling tool MODE-A (MOdel DErivation – Analytic approach) may be employed. Central component of this approach is a performance data base. For a set of relevant machines, it specifies the performance data of the SDL constructs (e.g. input, output, create) and the performance data of specific procedures called from the SDL processes. Examples are procedures provided by an implementation of an abstract data type or a procedure to perform a cyclic redundancy check. Provided that the data base contains the performance data for the required machines, the annotation of the respective MSCs can be done quickly without the need to actually implement and execute the SDL specification. The performance data for a particular machine in the data base can either be estimated based on performance data available for a comparable machine or based on measurements previously derived by MODE-M.

Note that our approach to model derivation is not necessarily limited to the automatic derivation of code from the given SDL specification. The approach can also be applied, if parts of the SDL specification are hand coded or different code generators are applied to generate code for different parts of the SDL specification.

2.4 Performance Evaluation and Optimization – MOPS

Once the necessary performance data of the SDL specification are derived, the performance-relevant design decisions can be made. The design decisions comprise software and hardware issues as well as the mapping of the software on the parallel machine. This is supported by the model based optimization tool MOPS (Model based Optimization of Parallel Systems). The outline of MOPS is depicted in figure 3.

The major input to MOPS consists of

- the machine model, i.e. the formal specification of the available parallel machine, and
- the load model, i.e. the formal specification of the load imposed on the machine including the performance requirements of the load.

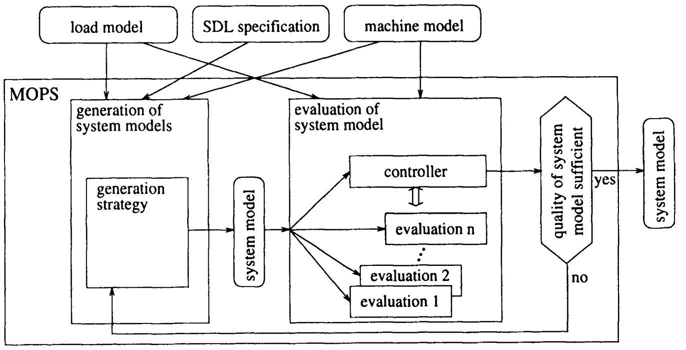


Figure 3 Model based optimization of parallel systems – MOPS

The machine model specifies a rather abstract machine. The abstract machine can be viewed as a parallel system that executes an abstract program. In our case, the abstract program is given by the HPMSC, described in detail in section 3.2. Note that the abstract machine also models the operating system. Thus, it supports process management including scheduling and load balancing. The open parameters of the machine which define the strategies for scheduling and load balancing are specified by the system model.

The tool computes the system model comprising the major design decisions concerning the static as well as dynamic aspects of the implementation. In general, the system model comprises the following design decisions:

- the static mapping of the code on the machines, i.e. the decision on which processor a specific SDL process can be executed,
- the dynamic load balancing strategy for the SDL processes,
- the dynamic scheduling strategy, e.g. the priorities of the processes, and
- the selection of implementation alternatives, possibly including hand-coded parts.

The MOPS tool consists of two main components, a component to generate system models (optimization method) and a component to evaluate given system models. The optimization method employed by MOPS is based on the subsequent improvement of an initial solution, i.e. the repeated improvement of an initial system model. Several optimization techniques supporting this approach are known from literature. The most important techniques are genetic algorithms, simulated annealing and tabu search. A comprehensive survey of these techniques can be found in (Reeves, 1993). The most appropriate optimization technique mainly depends on the time complexity of the evaluation component of MOPS.

The evaluation component performs the evaluation of given system models. In order to provide a design optimization tool that supports various optimization goals, we propose a flexible approach. In our approach, a set of basic predefined evaluation functions are provided to evaluate given system models. These evaluation functions may be complemented by user-defined evaluation functions. In addition, the interrelationship between the different (predefined and user-defined) evaluation functions and their influence on the main goal function, that quantifies the quality of a system model, is user-defined. Thus, different and possibly conflicting optimization goals may be pursued with different intensity. In addition, various optimization constraints may be specified and enforced. Examples of predefined evaluation functions are:

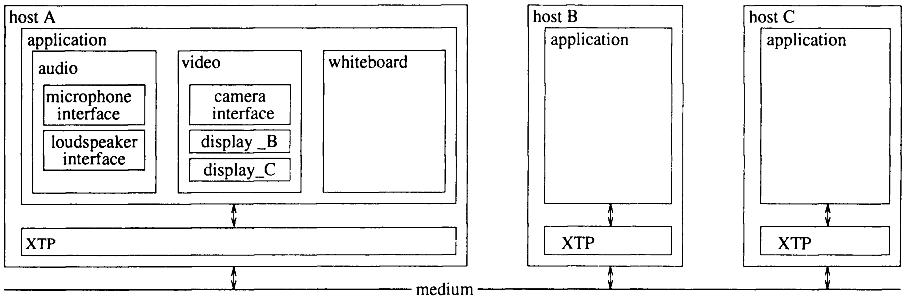


Figure 4 Outline of the multimedia application

(1) the evaluation of the (static) load per processor resulting from the load as specified by the load model (static bottleneck analysis),
(2) the evaluation of the (static) load per communication link (static bottleneck analysis),
(3) the evaluation of the response time of each use case, i.e. each MSC, under the assumption that no contention exists between the use cases, i.e. the use cases are imposed on the system sequentially rather than concurrently (zero load dynamic analysis),
(4) the evaluation of the response time of the use cases under the assumption that the maximum load, as specified by the load model, is imposed on the system concurrently (full load dynamic analysis).

The use of the last evaluation function is further detailed in section 3.4.

Several optimization algorithms have been implemented to compute a part of the design decisions relevant to configurable message passing systems (Mitschele-Thiel, 1993, Mitschele-Thiel and Dussa-Zieger, 1994, Schwehm and Walter, 1994).

3 APPLICATION OF THE DO-IT TOOLSET

3.1 Example: Multimedia Application

Our multimedia application is structured in two parts, the multimedia application itself and the underlying XTP protocol suite (release 4.0) to handle communication between the hosts. The structure of the multimedia system is depicted in figure 4. Three kinds of applications are supported: video, audio and whiteboard. We assume that data compression is employed for all applications. The functional requirements for transport connections are as follows: Audio and video require a non-confirmed connection-oriented transport service. The whiteboard application is handled by a confirmed connection-oriented transport service.

The performance requirements of the three types of multimedia applications differ considerably. An audio application requires a fixed number of small data packets. In order to keep the signal delay small, we require 22 packets per second. In addition, we require a maximum processing delay of 40 ms to send an audio packet and 50 ms to receive a packet.

Similarly, a video application also requires a fixed number of packets per time unit. However, the number of data packets transferred per time unit may be smaller. Typical throughput requirements for a video application vary from a single packet every ten seconds to 30 packets per second. For a multimedia conference with two or three participants we require a throughput of ten packets per second. In case of four participants, the required throughput is 5 packets per second. The delay of video signals is less critical than audio. We require a processing delay smaller than 200 ms for inbound and outbound traffic, respectively.

For the whiteboard application, we require a processing delay below 200 ms. The required throughput of the whiteboard application is two packets per second.

For a conference with two participants, two video connections, two audio connections (one for each direction) and one connection for the whiteboard application is required. In case a third participant enters the conference, three video connections are used altogether, one multicast connection to handle all outgoing video streams and one connection for each incoming video stream. The same holds for audio. For the whiteboard application, two connections are employed, one connection per remote participant.

3.2 Formal Specification of the Multimedia Application

Functional Specification with SDL

The structure of our SDL specification is given in figure 5. The octagons denote SDL processes. The solid lines denote communication, dashed lines model the creation of process instances. The SDL specification is structured in two blocks, one block for the application and one for XTP itself.

In the XTP block, separate SDL processes are used to handle the different applications (audio, video and whiteboard). In addition, inbound and outbound traffic is separated, i.e. handled by different SDL processes. Thus, a separate sender, receiver and encoder exists for each application. The XTP connections are managed by the context manager, which creates and terminates the process instances.

In the application block, separate SDL processes are used for incoming and outgoing data streams as well as for each application. An exception is the whiteboard application, which is handled by a single SDL process. Each inbound video stream requires a separate process instance. Similarly, each outbound video stream (each local camera) is handled by a separate process instance. A similar approach is taken for audio streams. When a new conference is started or a new participant enters the conference, the manager process in the application block creates the respective process instances and instructs the context manager in the XTP block to generate the corresponding process instances there.

Formal Specification of Performance Requirements with HPMSC

As already mentioned in section 2.2, we employ high-level performance MSCs (HPMSC), a notation based on use cases to formally specify, among others, the performance requirements of the system under development. A HPMSC is structured as a tree. The leaves of the tree refer to MSCs in most cases. The HPMSC employed to specify the performance requirements of our multimedia application is given in figure 6.

The syntax of HPMSCs is as follows: Four types of nodes exist, namely the root, MSC nodes, construction nodes and comment nodes. MSC nodes refer to an MSC. The referred MSCs are typically annotated with performance aspects. Four of the MSCs referred to in figure 6 are shown in figures 7 to 10.* Comment nodes are used to include informal text. Six different construction nodes exist. The construction nodes ALT, AND, PAR and SEQ are used to construct a larger tree from a set of subtrees. With exceptions, construction nodes have more than one child connected

*The notation employed by the MSCs is as follows: Each vertical axis denotes a process instance. A solid arrow models the transmission of a signal. A dashed arrow denotes the creation of a process instance.

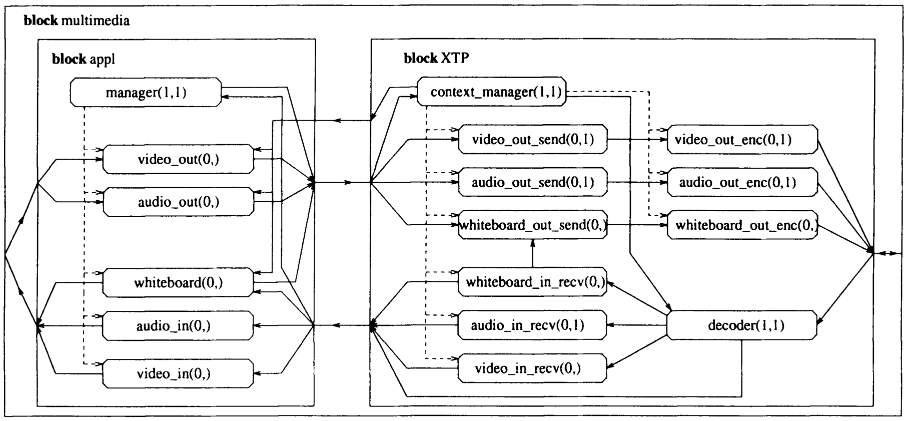


Figure 5 Structure of the SDL specification of the multimedia application

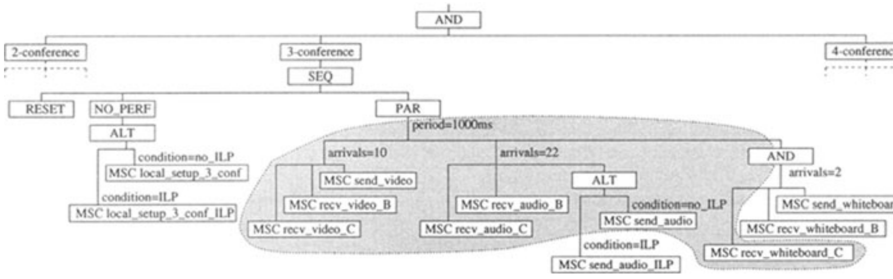


Figure 6 High-level performance MSC (HPMSC) to specify the performance requirements of the multimedia application

to it. Exceptions are the construction node RESET which does not have a child at all and the node NO_PERF which has exactly one child.

The semantics of the construction nodes is as follows: Each subtree in the HPMSC represents a load on the system with specific performance requirements. The PAR node denotes the case where the system under development has to be able to concurrently handle the load as given in each of the subtrees of the PAR node. Thus, the load is defined by the sum of the load specified by the subtrees. In the example given in figure 6, the PAR node is used to state that the various data streams (audio, video and whiteboard) are concurrently imposed on the sytem.

Similarly, the AND node also denotes the fact that the system has to be able to handle the load as given in each of the respective subtrees. However, different from the PAR node, the load as specified by the representative subtrees of the AND node is not imposed on the system concurrently. Rather, the load specified by each subtree has to be handled by the system one (subtree) at a time. In the example given in figure 6, the AND node is used to state that the system has to be able to handle each of the three types of conferences, i.e. with two, three or four participants. However, the system does not have to be able to handle all three types of conferences concurrently.

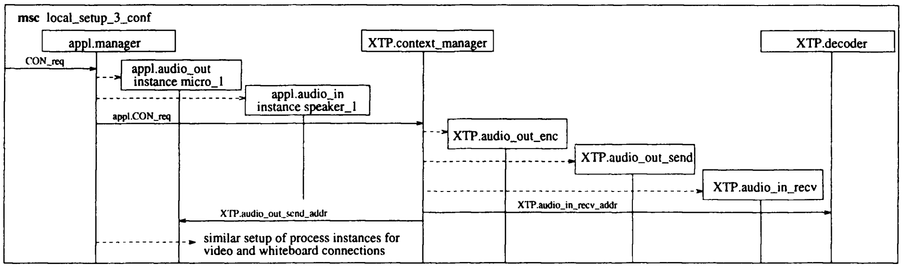


Figure 7 Local setup of a multimedia conference with three participants

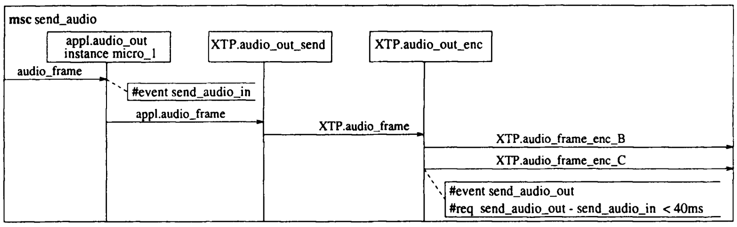


Figure 8 Sending of audio data

The SEQ node states that the system execution as specified by the subtrees is done sequentially, executing the subtrees from left to right.

The ALT node is used to express open design or implementation decisions. In this case, the selection of one of the alternatives is up to MOPS. Thus, MOPS implicitly selects the subtree in the HPMSC, which results in the best value of the overall optimization goal function. In other words, MOPS selects a subtree of the ALT node such that the resulting system model is optimal. In case that several ALT nodes are present in an HPMSC, MOPS decides on the best combination of all the alternatives.

In addition to the different types of nodes, attributes to the nodes can be specified. Each subtree of an ALT node is attributed with a condition. The conditions are evaluated statically before the performance evaluation is started. The variables of a condition are defined in the system model. The actual values of the variables are computed by the optimization algorithm employed by MOPS. In figure 6, an example of the use of an ALT node to express a design alternative for the audio transmission is given. In the example, the ALT node is used to express the design alternative where integrated layer processing (ILP) (Clark and Tennenhouse, 1990) is used to merge the two audio processes audio_out_send and audio_out_enc of the block XTP. It also shows how the condition attribute can be used to represent context dependences. In the figure, two different MSCs for the setup of the conference are given, one for the non-ILP design alternative and one for the case that ILP is employed.

Attributes important for performance evaluation are the attributes period and arrivals. The attribute period specifies the time period for which the performance evaluation is done. The arrivals attribute specifies the number of arrivals for the given MSC (or more specifically the number of its executions) in the given time period (as specified by the period attribute). Thus, the arrivals attribute in conjunction with the period attribute supports the specification of the load which the system under development is supposed to handle. This is

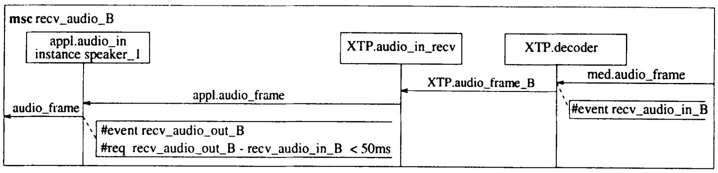


Figure 9 Reception of audio data frcm user B

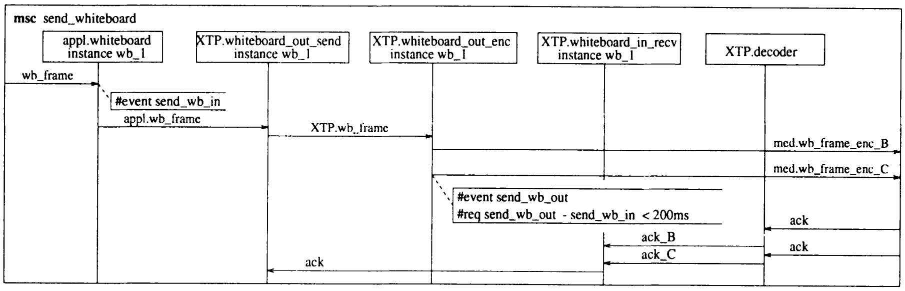

msc send_whiteboard
appl.whiteboard instance wb_1
XTP.whiteboard_out_send instance wb_1
XTP.whiteboard_out_enc instance wb_1
XTP.whiteboard_in_recv instance wb_1
XTP.decoder
wb_frame
#event send_wb_in
appl.wb_frame
XTP.wb_frame
med.wb_frame_enc_B
med.wb_frame_enc_C
#event send_wb_out
#req send_wb_out - send_wb_in < 200ms
ack
ack_B
ack_C
ack
ack

Figure 10 Transmission of a local update of the whiteboard

typically used to specify throughput requirements of the system. Both, the `period` and the `arrivals` attribute may be present with the node types `ALT`, `AND`, `PAR` and `SEQ`. The attributes are inherited by their subtrees. However, attributes may be redefined in a subtree.

Not necessarily each MSC (or even subtree containing a set of MSCs) in the HPMSC is directly relevant for the performance evaluation. Some MSCs may be given solely to initialize the system to a state where performance evaluation can be started. For example in communication systems, we are often solely interested in the performance of the data transmission and not so much in the connection establishment phase preceding the data transmission. Nevertheless, the connection must be established before data transmission can be started. For example, the appropriate process instances have to be created and initialized. In order to distinguish between subtrees relevant for the performance optimization and subtrees not relevant, the `NO_PERF` node has been introduced. `NO_PERF` excludes the subtree from any performance evaluation. The subtree of the `NO_PERF` node is solely used to initialize the system.

In a system, several performance evaluations are typically employed. As stated above, the results of the performance evaluation may depend on the state of the system from which a specific performance evaluation is started. In order to support the start of a performance evaluation at a defined state, the `RESET` node in conjunction with the `NO_PERF` node and a respective MSC to set the system into the appropriate initial state for performance evaluation, is employed. The `RESET` node puts the system in its initial state as specified in the SDL specification.

The MSCs referred to in the leaves of the HPMSC serve three purposes. First, they are used to specify the functional aspects relevant to performance evaluation, e.g. the creation of process instances prior to their use. An example of this is the MSC local_setup_3_conf given in figure 7.

Second, the MSCs are employed to specify response time requirements. Examples of these can be found in figures 8 through 10. For example, in figure 8 two events are defined, event send_audio_in and event send_audio_out. The response time requirement itself is given in the `req` clause. It states that the time required to process an outgoing signal has to be smaller than 40 ms.

Third, the MSCs are employed to specify performance data. However, note that performance data are not part of the requirements specification. Rather, they are derived by the MODE component. For this reason, discussion on the specification of performance data is deferred to section 3.3.

Machine Model

The parallel target machine for our multimedia application is a Sun Sparc 20 station with four processors. We assume that three of the processors are solely dedicated to handle the multimedia application as specified above. All other processes on the workstation are assumed to be run on the fourth processor.†

As a result, the machine model defines three identical processors available to handle the load as specified by the HPMSC given in figure 6. The abstract parallel machine comprises a small set of abstract instructions, namely

- `create` and `terminate` to set up and terminate a process instance.
- `comp` to execute a computation on an abstract processor,
- `input` and `output` to support communication between the process instances,
- `set_timer` to induce a special timer signal in the system at a specified time.

The abstract instructions directly correspond to the activities specified with the MSCs. As noted before, the abstract instructions return the cost of their execution on the different resources. In this simple example the only resources considered are the processors. Other possible resources could be communication links and storage devices.

In addition to the abstract instructions corresponding to the MSC activities, instructions that implement the functions of the (abstract) operating system are provided by the machine model. This includes a function to implement the scheduler for the MSCs. In addition, functions are provided to fetch the abstract MSC instructions described above and to read information from the system model.

For our example, we assume that the three processors are clustered to dynamic load sharing groups. Each SDL process is statically assigned to one of the load sharing groups. Each load sharing group employs its own scheduler. Thus, a given process instance can be dynamically scheduled on each of the processors of the load sharing group. Migration of processes between load sharing groups is not allowed.

3.3 Application of MODE

The purpose of the MODE tools is to derive performance data for the application. As outlined in section 2.2, performance data either specify the actual cost to execute the entities described with the MSCs or specify the parameters from which the respective cost can be derived. The DO-IT toolbox supports both approaches. The mapping of parameterized cost to actual cost is done by the abstract instructions defined by the machine model. In the example, actual cost are used to specify computation cost. Parameters are employed for the specification of cost related to communication. Thus, the actual cost for communication may depend on the location of the communicating processes and is computed after this has been decided. The physical location of

† A mixture of the load is possible but is not described here for space limitations.

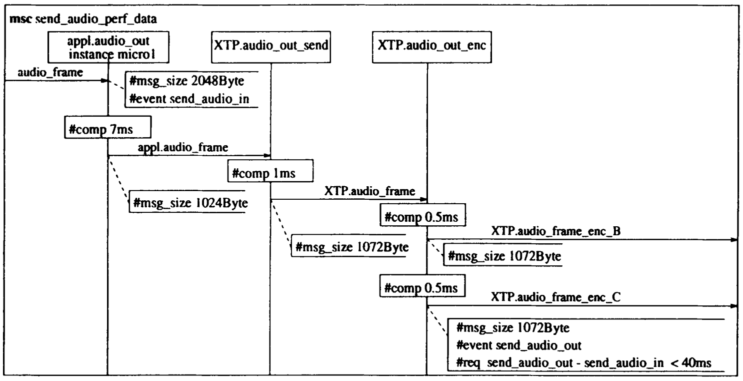


Figure 11 Annotation of the MSC send_audio with performance data

process instances is either statically defined by the system model or decided at runtime during the creation of process instances. This is the case if several process instances are created from an SDL process and dynamic or semi-dynamic load balancing is employed.

As explained in section 2.3, MODE derives the performance data for each MSC specified by the HPMSC given in figure 6. In addition, MODE performs the back annotation of the MSCs with the derived performance data.

An example for the specification of performance data in the MSC send_audio, as specified in figure 8, is shown in figure 11. In the figure, the computation cost are given in the `comp` clause. Communication cost are derived from the size of the messages which is given in the `msg_size` clause.

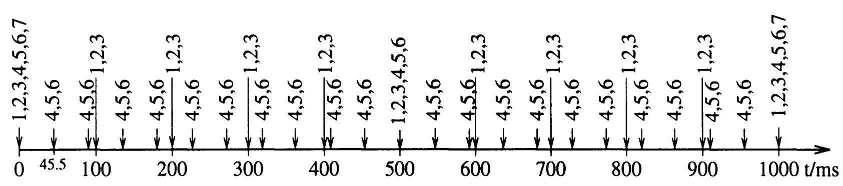
3.4 Application of MOPS

As described in section 2.4, the input to MOPS mainly consists of the abstract specification of the machine and the load. Open parameters of the machine and the load are defined by the system model. In addition, the system model defines the design decisions related to the mapping of the load on the machine. For space limitations, we concentrate on the evaluation of a given system model and its prerequisites. We describe in more detail how the evaluation of the dynamic behavior of the system under full load is done. This refers to the fourth evaluation function given in section 2.4.

System Model

During the optimization process of MOPS, we assume that at some state of the optimization process, the following system model has been computed:

- two load sharing groups, S1 and S2, are used,
- processor P1 is assigned to the load sharing group S1; P2 and P3 are assigned to S2,
- all SDL processes related to audio including the XTP decoder are assigned to S1, the remaining processes are assigned to S2,
- the priorities of the SDL processes assigned to S1 – specified in decreasing order of their priority – are as follows: XTP.audio_out_enc, XTP.audio_out_send, appl.audio_out, appl.audio_in, XTP.audio_in_recv, XTP.decoder,



Legend: 1: MSC send_video	4: MSC send_audio	7: MSC recv_whiteboard_C
2: MSC recv_video_B	5: MSC recv_audio_B	
3: MSC recv_video_C	6: MSC recv_audio_C	

Figure 12 Temporal order of the signal arrivals for the specified load scenario

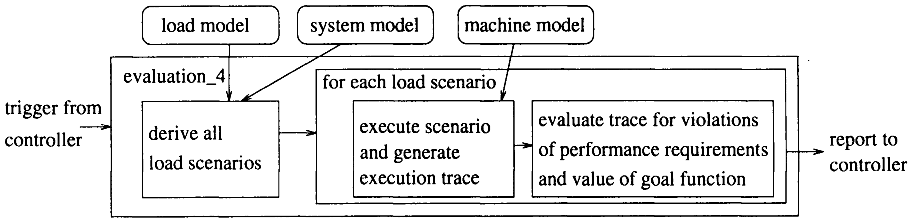

load model
system model
machine model
evaluation_4
trigger from controller
derive all load scenarios
for each load scenario
execute scenario and generate execution trace
evaluate trace for violations of performance requirements and value of goal function
report to controller

Figure 13 Evaluation of the dynamic behavior of a system model under full load

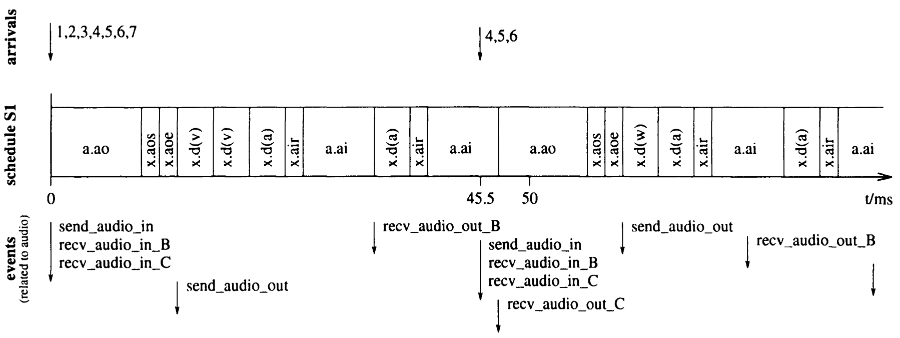
- the priorities of the remaining SDL processes assigned to S2 are as follows: appl.manager, XTP.context_manager, XTP.whiteboard_out_enc, XTP.whiteboard_out_send, appl.whiteboard, XTP.whiteboard_in_recv, XTP.video_out_enc, XTP.video_out_send, appl.video_out, appl.video_in, XTP.video_in_recv,
- the ILP option is turned off.

Example of the Evaluation of the System Model

Before the evaluation function is described, we introduce *load scenarios*. A load scenario represents an example of the load imposed on the system within a given time period. For example, the load model defined by the HPMSC given in figure 6 can be divided in three parts according to the number of participants of the multimedia conference. The detailed subtree specifying the conference with three participants defines three load scenarios. This is because of the AND node in the subtree that distinguishes three load cases for the whiteboard application. In the figure, the MSCs belonging to the load scenario used in the following example are marked by the grey background.

A load scenario implicitly defines a temporal order on a set of input signals. The temporal order of the arrivals derived from the load scenario marked in figure 6 is depicted in figure 12. Note that in conjunction with the machine and the system model, the temporal order on the input signals fully defines the execution order (schedule) of the abstract machine.

In figure 13, the evaluation function to evaluate the dynamic behavior of the system model under full load is outlined. The evaluation is done in two phases. First, the different load scenarios are derived from the load model. Second, the system model is evaluated for each load scenario. For our specific evaluation function, the evaluation is done by abstract execution of the load scenarios on the abstract machine according to the guidelines provided by the system model.



Legend of the scheduled process instances:

a.ai: appl.audio_in	x.air: XTP.audio_in_recv	x.d(a): XTP.decoder(audio data)
a.ao: appl.audio_out	x.aoe: XTP.audio_out_enc	x.d(v): XTP.decoder(video data)
	x.aos: XTP.audio_out_send	x.d(w): XTP.decoder(whiteboard data)

Figure 14 Schedule of the load sharing group S1

The schedule of the load sharing group S1 for the example load scenario is depicted in figure 14. Note that for the sake of clear presentation, the cost are not given in their exact scale. Also note that computation cost related to communication between the process instances are not explicitly shown. Instead, they are added to the computation cost of the process instances.

During the abstract execution, events defined by the MSCs are traced and checked against the requirements also defined by the MSCs. The events related to audio traffic are depicted in figure 14. In addition to the evaluation of the events, the value of the optimization goal function is derived.

4 CONCLUSIONS

The DO-IT toolbox is a joint project of the University of Erlangen and the Technical University of Cottbus. The goal of the DO-IT toolbox is to support

- the automatic derivation of the performance data incurred with the implementation of the components of an SDL specification,
- the analysis of SDL specifications for performance bottlenecks,
- the synthesis of an optimized system design deciding the issues related to software, hardware and the mapping of the software on the hardware, and
- the implementation of the design decisions.

Central to the early integration of performance issues in the design process is the extension of MSCs to formally specify performance-relevant information.

In the paper, we have concentrated on the first and the third issue. We have shown how the DO-IT toolbox can be employed to derive the implementation design of a multimedia application. Designing such an application, many conflicting performance requirements have to be met and the best compromise has to be found. Especially with the implementation on parallel systems, a large number of design parameters need to be decided on. This results in an enormous design space. In the paper, an approach for the automatic search of the design space has been described. In addition, we have shown how different classes of design decisions can be included in the automatic search process.

ACKNOWLEDGEMENTS

Many people have contributed to the design of the DO-IT toolbox. Thanks for their support of the project and for numerous discussions go to H. König, U. Herzog, W. Dulz, N. Faltin, L. Lambert and F. Lemmen.

REFERENCES

D.D. Clark, D.L. Tennenhouse. (1990). Architectural considerations for a new generation of protocols. ACM SIGCOMM, p. 200-208.

P. Dauphin, R. Hofmann, R. Klar, B. Mohr, A. Quick, M. Siegle, F. Sötz. (1994) ZM4/SIMPLE: a General Approach to Performance-Measurement and -Evaluation of Distributed Systems. Readings in Distributed Computing Systems, T.L. Casavant, M. Singhal (Ed.), IEEE Computer Society Press.

P. Dauphin, W. Dulz, F. Lemmen. (1995) Specification-driven Performance Monitoring of SDL/MSC-specified Protocols. Proc. 8th Int. Workshop on Protocol Test Systems, A. Cavalli, S. Budkowski (Ed.).

R. Hofmann, R. Klar, B. Mohr, A. Quick, M. Siegle. (1994) Distributed Performance Monitoring: Methods, Tools and Applications. IEEE Transactions on Parallel and Distributed Systems, 5(6).

ITU-T. (1993) Z.100, Specification and Description Language (SDL). ITU.

ITU-T. (1993a) Z.120, Message Sequence Chart. ITU.

A. Mitschele-Thiel. (1993) Automatic Configuration and Optimization of Parallel Transputer Applications. Transputer Applications and Systems '93, R. Grebe et al. (Ed.), vol. 2, IOS Press.

A. Mitschele-Thiel, K. Dussa-Zieger. (1994) Near-Optimal Compile-Time Scheduling and Configuration of Parallel Systems. Proc. of the 1994 World Transputer Congress, Lake Como, Italy, IOS Press.

A. Mitschele-Thiel. (1996) Methodology and Tools for the Development of High Performance Parallel Systems with SDL/MSCs. Software Engineering for Parallel and Distributed Systems, I. Jelly, I. Gorton, P. Croll (Ed.), Chapman & Hall.

C.R. Reeves (Ed.). (1993) Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publications, Oxford.

M. Schwehm, T. Walter. (1994) Mapping and Scheduling by Genetic Algorithms, Parallel Processing: CONPAR'94–VAPP VI, Third Joint Int. Conf. Vector and Parallel Processing, Lecture Notes in Computer Science 854, Springer-Verlag.

W.T. Strayer, B.J. Dempsey, A.C. Weaver. (1992) XTP: The Xpress transfer protocol. Addison Wesley.

Telelogic Malmö AB. (1995) SDT 3.0 User's Guide, SDT 3.0 Reference Manual.

Verilog. (1994) GEODE – Technical Presentation.