Model-Based Monitoring in Large-Scale Distributed Systems

Diploma Thesis

Carsten Reimann

Chemnitz University of Technology
Faculty of Computer Science
Operating System Group

Advisors: Prof. Dr. Winfried Kalfa
Dr. Sven Graupner
Model-Based Monitoring in Large-Scale Distributed Systems
Chemnitz University of Technology, Faculty of Computer Science, Operation System Group
Diploma Thesis

By: Carsten Reimann
Born 05/07/1978 in Zwickau, Germany

Issued: 03/01/2002
Published: 05/31/2002
**Task**

The goal of this work is to define a model description of a chosen distributed application running in the CLIC environment, represent this model with its elements and parameters in the database built in the previous project and build a monitoring infrastructure which allows to constantly calibrate model parameters such that at any time a realistic picture of the application’s behavior can be obtained. Another goal is to collect statistics (traces) of how model parameters have evolved over time. Experiments conclude the work demonstrating the concept and implementation.

Following steps are proposed:

1. Choosing a distributed application environment
2. Identifying components and links with associated parameters representing the behavior of that application
3. Identifying appropriate metrics to express behavioral characteristics
4. Mapping the model into a database schema
5. External presentation of model information
6. Sensor infrastructure
7. Experimentation in the CLIC
Abstract

Monitoring remains an important problem in computer science. This thesis describes which monitor information is needed to analyze distributed service environments. This thesis also describes how to get these information and how to store them in a monitoring database. The resulting model is used to describe a distributed media content environment and a simulation system that runs on the CLIC helps to generate measurements as in real systems.
## Contents

1. **Introduction** .................................................................................................................. 9  
   1.1 Scenario .......................................................................................................................... 9  
   1.2 Goal .................................................................................................................................. 9  

2. **Precursory Work** ...........................................................................................................11  
   2.1 Data Structure ....................................................................................................................11  
   2.2 Interface ............................................................................................................................12  
   2.3 Browser Adapter ..................................................................................................................13  
   2.4 Overview ...........................................................................................................................13  

3. **Modeling Services** ....................................................................................................... 15  
   3.1 Introduction .......................................................................................................................15  
   3.2 Basic Services ....................................................................................................................15  
   3.3 Characteristic Parameters of Basic Services .......................................................................16  
   3.4 Distributed Services .........................................................................................................17  
   3.5 Characteristic Parameters of Distributed Services .............................................................18  
   3.6 Summary ............................................................................................................................19  
   3.7 Getting Characteristic Parameters ....................................................................................19  
   3.7.1 Measurements ..............................................................................................................19  
   3.7.2 Evaluating Service Parameters ....................................................................................20  
   3.7.3 Getting Measurements ...................................................................................................20  
   3.7.4 Measurement Failure ......................................................................................................21  
   3.8 Sensors ..............................................................................................................................21  
   3.8.1 Location .........................................................................................................................21  
   3.8.2 Activity ..........................................................................................................................22  
   3.9 Database Storage ..............................................................................................................23  
   3.10 XML Representation .......................................................................................................24  
   3.10.1 Current State ..............................................................................................................25  
   3.10.2 Trace ............................................................................................................................27  

4. **Distributed Media Content** ........................................................................................ 29  
   4.1 Overview ...........................................................................................................................29  
   4.2 System Model ....................................................................................................................29  
   4.2.1 System Characteristics .................................................................................................29  
   4.2.2 Customer Behavior ......................................................................................................30  
   4.2.3 System Structure ..........................................................................................................30  
   4.2.4 Typical Parameters of Media Contents ..........................................................................31
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.5 Typical Customer Behavior</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Service Models</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1 Sensor</td>
<td>32</td>
</tr>
<tr>
<td>4.3.2 Summary</td>
<td>33</td>
</tr>
<tr>
<td>4.3.3 Database Storage</td>
<td>35</td>
</tr>
<tr>
<td>4.3.4 XML Representation</td>
<td>36</td>
</tr>
<tr>
<td>5 Simulation with Service Models</td>
<td>37</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>37</td>
</tr>
<tr>
<td>5.1.1 Model Structure</td>
<td>37</td>
</tr>
<tr>
<td>5.1.2 Policy</td>
<td>37</td>
</tr>
<tr>
<td>5.1.3 Requirements</td>
<td>38</td>
</tr>
<tr>
<td>5.2 Methods</td>
<td>38</td>
</tr>
<tr>
<td>5.3 Synchronization</td>
<td>39</td>
</tr>
<tr>
<td>6 Implementation</td>
<td>41</td>
</tr>
<tr>
<td>6.1 Simulation</td>
<td>41</td>
</tr>
<tr>
<td>6.1.1 Processes</td>
<td>41</td>
</tr>
<tr>
<td>6.1.2 Configuration</td>
<td>47</td>
</tr>
<tr>
<td>6.2 Browser Adapter</td>
<td>52</td>
</tr>
<tr>
<td>6.2.1 Overview</td>
<td>53</td>
</tr>
<tr>
<td>6.2.2 XML-Message</td>
<td>53</td>
</tr>
<tr>
<td>7 Examples</td>
<td>55</td>
</tr>
<tr>
<td>7.1 A Cause for Media Centers</td>
<td>55</td>
</tr>
<tr>
<td>7.1.1 Service Model Configuration</td>
<td>55</td>
</tr>
<tr>
<td>7.1.2 Results</td>
<td>58</td>
</tr>
<tr>
<td>7.1.3 Conclusion</td>
<td>61</td>
</tr>
<tr>
<td>7.2 Preloading Media Content</td>
<td>61</td>
</tr>
<tr>
<td>7.2.1 Service Model Configuration</td>
<td>61</td>
</tr>
<tr>
<td>7.2.2 Results</td>
<td>62</td>
</tr>
<tr>
<td>7.2.3 Conclusion</td>
<td>64</td>
</tr>
<tr>
<td>8 Conclusion</td>
<td>65</td>
</tr>
<tr>
<td>8.1 Possibilities for the Future</td>
<td>65</td>
</tr>
<tr>
<td>8.2 Related Work</td>
<td>66</td>
</tr>
<tr>
<td>Acronyms</td>
<td>67</td>
</tr>
<tr>
<td>Figures</td>
<td>69</td>
</tr>
<tr>
<td>Bibliography</td>
<td>71</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Scenario

Monitoring remains an important problem in computer systems. Research at HP Labs anticipates that computer systems and their management systems will become much larger in future. Existing monitoring techniques such as those used in products like HP OpenView are assumed to reach their limits for several reasons. They collect too fine-grained data such as utilization of networks, machines, storage and other hardware components.

The goal of a project at HP Labs called self-organizing services is to better understand system behavior at higher-order software and service layers. Since hardware infrastructure is shared among different applications and services, it is hard to derive application- and service-level behavior from the collected monitoring data. Though techniques have been developed for deriving and condensing application shares from aggregated load in a bottom-up fashion, these approaches are complicated and only applicable in certain areas.

1.2 Goal

The goal of this work is to define a model at application layer of distributed systems. It attempts to find out which metrics can be identified for individual services that characterize their behavior and how these metrics can be obtained from monitoring a real system. These monitoring data should be stored into the database built in the previous project [XMDB]. Therefore the model must be mapped into a database schema to enable adjustment of model parameters by constantly monitoring them later in the real application system.

The measured values need to be analyzed in two ways. First, the “statistical current” model description with a set of model parameter obtained by measures in the real system. Second, it contains statistical traces showing how model parameters have changed over time. The database schemata for the model and for the history traces should be exposed in form of separate downloadable XML [XML] documents outside the database allowing the discoverability of the data structure.

The model information must be made externally accessible by defining queries over these data sets. The existing XML and browser interface of the database should be used for this purpose and possibly be extended.
Another goal is to use the model by choosing a distributed application for monitoring. The application should use multiple components that communicate among each other.
2 Precursory Work

The precursory work was a database build by Koenig, R. and Reimann, C. It allows to store monitoring data into hierarchically structured measurement tables. The database operates with an XML-interface for data access and data manipulation. It uses HTTP for communication and is accessible by standard browsers like Internet-Explorer and Netscape Communicator. The previous work is completely described in [XMDB].

2.1 Data Structure

All monitoring data can be structured hierarchically for storage in the database. This structure is comparable to the organization of file systems. There are folders to organize the structure and there are parameter tables as containers for measured values. A simple example of their use is shown in Figure 2-1. It contains the organization of the CLIC (“Chemnitzer Linux Cluster”): There are different machines (e.g. “clic2f23”) and they each have data tables for measuring values, in this example “memory” and “cpu”.

```
CLIC
+clic2f23
| +memory
| +cpu
+clic3i31
| +memory
| +cpu
```

Figure 2-1: Simple structure in database

Every item in the tree structure has a parameter that shows its actual state. There are 3 different possible states:

- **“Active”** The data table is ready for measurement values
- **“Inactive”** The data table is temporarily not ready for measurement values
- **“Closed”** Measurement is complete and only reading access is possible

Measurement data to be stored into a data table consist of two values. One is the timestamp and the other is the dedicated data value. The database is able to store different data types into the data tables. Possible types are Integer-, Float- and String-Values. A few typical examples are shown in Figure 2-2.
2 Precursory Work

Real-values:

<table>
<thead>
<tr>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002-01-23 15:21:02</td>
<td>15.6</td>
</tr>
<tr>
<td>2002-01-23 15:21:04</td>
<td>24.1</td>
</tr>
<tr>
<td>2002-01-23 15:21:05</td>
<td>20.9</td>
</tr>
<tr>
<td>2002-01-23 15:21:07</td>
<td>19.2</td>
</tr>
<tr>
<td>2002-01-23 15:21:10</td>
<td>11.4</td>
</tr>
<tr>
<td>2002-01-23 15:21:12</td>
<td>2.9</td>
</tr>
</tbody>
</table>

String-values:

<table>
<thead>
<tr>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002-01-24 11:32:41</td>
<td>WAIT</td>
</tr>
<tr>
<td>2002-01-24 11:32:43</td>
<td>RECEIVE</td>
</tr>
<tr>
<td>2002-01-24 11:32:44</td>
<td>COMPUTE</td>
</tr>
<tr>
<td>2002-01-24 11:32:45</td>
<td>WAIT</td>
</tr>
<tr>
<td>2002-01-24 11:32:46</td>
<td>WAIT</td>
</tr>
<tr>
<td>2002-01-24 11:32:48</td>
<td>SEND</td>
</tr>
</tbody>
</table>

Figure 2-2: Parameter table examples

2.2 Interface

All communications with the database (e.g. insert records or manipulate tree structure) are though XML. That means that all commands to the database are in the form of XML-documents. The exchange of these documents is realized with the servlet technology and uses HTTP as transport protocol. HP Labs offered this messaging system.

There are 3 types of command messages:

- **Queries**  
  Used for getting stored data values
- **Manipulations**  
  Used for insertion, changing or deletion of data values
- **Management**  
  Used to manage the hierarchical structure (e.g. create tables)

Figure 2-3 shows a sample message for a data query from the “cpu”-measurements of the “clic2f23” machine in the “CLIC”. All other command messages are built similarly.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<body>
  <dataQuery>
    <tableselect>
      <sub name="root" />
      <sub name="CLIC" />
      <sub name="clic2f23" />
      <sub name="cpu" />
    </tableselect>
    <interval>
      <begin dateTime="2002-01-23 15:00:00" />
      <end dateTime="2002-01-23 15:30:00" />
    </interval>
  </dataQuery>
</body>
```

Figure 2-3: Message for data query
2.3 Browser Adapter

The database also offers a user interface, which can be used with standard browsers. This interface, a so-called “Browser Adapter”, communicates through the XML interface described in section 2.2 and is also based on servlet technology. This user interface offers a comfortable use of the database and presentation of monitoring data. The presentation of monitoring data can be textual and also graphical by using SVG, a language to describe two-dimensional graphics in XML.

2.4 Overview

Figure 2-4 shows the main components of the database and how they cooperate. Tomcat 3.3 is used as a HTTP-server and servlet engine. And PostgreSQL 7.1.2 is used as a relational database.

Figure 2-4: Main components of the database
3 Modeling Services

3.1 Introduction

Typical distributed systems consist of different activities, which are communicating with each other [MVS]. For the analysis of distributed systems we need a model that represents the whole system. The model should support performance analysis of the system. That means to find out: Where are the bottlenecks? And, which components are rarely used? Quasi: Analyze a system for an optimization. A current performance model of distributed systems is the queuing network model. It supports the needed analysis of distributed systems. Therefore this model is the fundament of the following service model.

3.2 Basic Services

Services are activities, which process certain jobs. They are hosted on an infrastructure called server, this may be a simple computer, a multi processor machine, a cluster system and the like. Several services can be hosted on such a server:

![Diagram of services on a server]

Figure 3-1: Services on a server

The general procedure of a service process is: Waiting for a job, receiving job information from a client, processing the job, sending the result of the job back to the client and waiting for next job. The client is another process, which is hosted on the same server or on a remote system (see chapter 3.4.) Jobs can be e.g. transactions on a database, HTML-requests to a web-server or media requests to a media content server in a VoD system.
If we look at a web-server on a Linux machine, the server will be the computer on which the web-server process (e.g. Apache [AWS]) runs. The service process will be the web-server process. The states of this service process are shown in Figure 3-3. The web-server sleeps until a client requests a HTML-page. After the request the server provides the requested HTML-page and the web-server sends it then to the client.

3.3 Characteristic Parameters of Basic Services

A basic service can be treated like a single queuing station in a queuing network [PCCS], [MLR]. Important questions are: How many jobs can the service process? How many jobs gets the service? These are parameters, which depends on a period of time:

We can define the following parameters:

\[ \lambda(t, \Delta t) : \text{The number of incoming jobs per second (request rate) in a particular time period} \]
3.4 Distributed Services

In the previous chapter we analyzed basic services. In practice, many systems consist of multiple, distributed services [MDS]. Distributed services means that there are different services on different servers. These services depend on each other; one service needs other services for the completion of a job. Services are in relation with each other, they form a kind of network. Distribution also means that several services, on different servers, offer the same scope of function. Which service is used can be chosen by the client or a distributing activity. This needs a slight change in the service state model:

\[ \mu(t, \Delta t) : \text{The mean number of jobs per second, which the service is able to process} \]

\[ \frac{1}{\mu(t, \Delta t)} : \text{The mean time to process one job} \]

\[ \rho(t, \Delta t) = \frac{\lambda(t, \Delta t)}{\mu(t, \Delta t)} : \text{The utilization of a service at a special time period} \]

\[ \lambda = 5 \text{ } \frac{1}{s} \quad \mu = 100 \text{ } \frac{1}{s} \]

\[ p = \frac{\lambda}{\mu} = 0.05 \]

Figure 3-5: A simple example of characteristic parameters

Figure 3-6: States of distributed service processes
3.5 Characteristic Parameters of Distributed Services

Let us take a look at a simple scenario on Figure 3-7, which consists of three service processes on three servers. A LAN connects them. Service process 1 is a web-server, which gets jobs (HTTP-requests) from web-browsers. Service processes 2 and 3 are database processes the web-server accesses. The sequence may be: The web-server gets a request from a web-browser, reads values from a database and sends the result back to the web-browser. The web-server gets requests with rate $\lambda_{0,1}$ and needs data from service process 2 with rate $\lambda_{1,2}$ and from service process 3 with rate $\lambda_{1,3}$. It must be pointed out that all rates depend on a special time period $(t, \Delta t)$.

![Figure 3-7: Scenario for distributed services](image)

The network created by distributed services can be analyzed with queuing networks [ALVS], [ICPE], [MLR], [PCCS]. Thus we can generally define the following parameters for distributed services:

- $\lambda_{0,n}(t, \Delta t)$: Request rate to service process $n$ from an external client
- $\lambda_{n,m}(t, \Delta t)$: Request rate from service process $n$ to service process $m$
- $\mu_n(t, \Delta t)$: Capacity of a service process $n$
- $\frac{1}{\mu_n(t, \Delta t)}$: Mean time to process one job at service process $n$
- $\rho_n(t, \Delta t) = \frac{\sum m \lambda_{m,n}(t, \Delta t)}{\mu_n(t, \Delta t)}$: Utilization of service process $n$
Additionally, it must be noted that requests to remote services take time and generate network load. This time results from the summated time for communication and the time waiting for the result. Thus we can additionally define the following parameters:

\[ \omega_{n,m}(t,\Delta t) : \text{Mean time one request takes from service process } n \text{ to service process } m \]

\[ \eta_{n,m}(t,\Delta t) : \text{Mean network load in KB caused by one request from service process } n \text{ to service process } m \]

### 3.6 Summary

We use a kind of queuing networks for the analysis of distributed service environments. With the queuing networks we have a model to describe the behavior of distributed services with stochastic parameters. Queuing networks do not contain any parameters of communication expanse. That is why we added the parameters \( \omega_{n,m}(t,\Delta t) \) and \( \eta_{n,m}(t,\Delta t) \). How all named parameters of this model can be measured is described in chapter 3.7.

### 3.7 Getting Characteristic Parameters

#### 3.7.1 Measurements

In this section the measurements are defined, which can be derived from real services to evaluate the service parameters [MLR]:

\[ \Delta t : \text{Period of time the measurement runs} \]

\[ r_n(\Delta t) : \text{Number of requests to service process } n \text{ in period } \Delta t \]

\[ u_n(\Delta t) : \text{Time used for processing all requests at service process } n \text{ in period } \Delta t \]

\[ \text{this includes the waiting time for other service processes } w_{n,m}(\Delta t) \]

\[ r_{n,m}(\Delta t) : \text{Number of requests from service process } n \text{ to service process } m \text{ in period } \Delta t \]

\[ w_{n,m}(\Delta t) : \text{Time service process } n \text{ waits for service process } m \text{ in period } \Delta t \]
$n_{n,m}(\Delta t)$: Summated network load produced by requests from service process $n$ to service process $m$ in period $\Delta t$

### 3.7.2 Evaluating Service Parameters

For evaluating characteristic service parameters, the following formulas can be used:

- $t$: Time when measurement period ends

\[
\lambda_n(t, \Delta t) = \frac{r_n(\Delta t)}{\Delta t}
\]

\[
\frac{1}{\mu_n(t, \Delta t)} = \frac{u_n(\Delta t)}{r_n(\Delta t)}
\]

\[
\lambda_{n,m}(t, \Delta t) = \frac{r_{n,m}(\Delta t)}{\Delta t}
\]

\[
\omega_{n,m}(t, \Delta t) = \frac{w_{n,m}(\Delta t)}{r_{n,m}(\Delta t)}
\]

\[
\eta_{n,m}(t, \Delta t) = \frac{n_{n,m}(\Delta t)}{r_{n,m}(\Delta t)}
\]

### 3.7.3 Getting Measurements

In order to get the measurements we need a kind of sensor that collects the data. This sensor could be embedded into the service process or in the environment of the service process. In any event, the data the sensor collects are the same. Let us review the behavior of a service (Figure 3-6): The service is waiting for jobs, processes jobs and sends jobs to other services. The sensor could take note of every state transition and counts the received and sent bytes.
For jobs the service receives by itself, the sensor does not have to count the received and sent bytes, because the sensor of the service, which sends the job, will do that already.

3.7.4 Measurement Failure

A sensor is an activity, thus it needs resources like processor time or network capacity for communicating with the database. So the sensor falsifies the measurements, e.g. the processing time of a request or something else. We assume that this influence is very slight, thus we will not include it in our considerations.

The other kind of measurement failures is that some requests are still in process when a measurement period is already finished. Thus there will be a failure in the processing time measures. This influence is also very lowly, because the measurement period is usually much longer than a request processing time. Therefore, those influences will not be apart of our considerations, either.

3.8 Sensors

3.8.1 Location

The location of the sensor may vary, it can be:

- Integrated in the service process
- Another process on the same server
- Another process on a server next to the service process
The first location, integrated in the service process, is very suitable, because you direct have access to all information you will need. You can do this by changing the source code of the service process or by changing libraries used by the service.

If the location is another process hosted by the same server, it must be a process that has access to the process control in order to get needed information. That means the process needs system privileges provided the host operating system supports process privileges. Alternatively, the sensor could read log-files from the service process to get the information.

For other processes on a different server the only way to get the information it needs is to tap the communication between service processes from the communication media. This creates many problems in identifying service processes if there are more than one hosted by the server. But you do not need to change any source code and you can have measures without influence of an internal sensor (see 3.7.4).

### 3.8.2 Activity

The sensor gets the information of the transition events. This includes the time of the transition and the new state of the service. The sensor gets also information about the expense of communication. The information sequence will be:

![Diagram of information sequence of the sensors](image)

The collected information could be completely stored or summarized. By storing every single event the storage space of the sensor will overflow quickly provided the event rate is very high. So it is absolutely necessary that the information must be summarized. This can be easily...
realized by counting every event type using a separate variable. The number of remote job events must be separated for each remote service, which is used. Now we have all needed measurements:

\[ \Delta t = \text{Time since the counting variables were zeroed} \]

\[ r_n(\Delta t) = \text{Number of Event<Incoming job>} \]

\[ u_n(\Delta t) = \text{Summated time between Event<Incoming job> and Event<Job complete>} \]

\[ r_{n,m}(\Delta t) = \text{Number of Events<Remote job> by using service } m \]

\[ w_{n,m}(\Delta t) = \text{Summated time between Event<Remote job> and Event<Remote job complete> by using service } m \]

\[ n_{n,m}(\Delta t) = \text{Number of bytes transmitted to and from service } m \]

The measured data should be stored in the database. This can be realized by inserting all collected values into the database after a time period \( \Delta t \). After that, all variables will be zeroed for the next period. Every sensor is directly connected to the database and works independently from all other sensors. So we do not need a central instance for database access.

This distribution of sensors has an important disadvantage: Every measurement value depends on a point of time. But the time in such a system may differ from unit to unit. So there is a need for synchronization. Time synchronization is a known problem in distributed systems and there are some possible solutions like Cristian’s Algorithm or the Berkeley-Algorithm they are described in [MBS].

### 3.9 Database Storage

The database is able to store structures and also measurement values. So we can store the structure of the distributed service environment with all measured data at once. To store the structure of the service environment you must know their components. It consists of different servers with a link for communication and every server is a host for multiple services. Firstly, we start with identifying the whole environment by a folder in the database. This should be the top-folder. The next step is to present every server by a folder with the environment-folder as
“father”. The same procedure needs to be done with all services. The result of these steps may be as shown in Figure 3-10. But there are also other hierarchies possible, e.g. which contain subnets or the like.

![Diagram of possible database structure](image)

**Figure 3-10: Possible database structure**

Now we can integrate the measurement value tables. There are two measurements that only belong to the service itself and should be directly stored in the service folder: $r_n(\Delta t)$ and $u_n(\Delta t)$. The other values are related to other (remote) services and should be associated with them. The resulting table structure is shown in Figure 3-11.

![Diagram of possible representation of the three-server system](image)

**Figure 3-11: Possible representation of the three-server system**

### 3.10 XML Representation

The XML format should be used for external representation (see 1.2). It should be possible to extract the current state and also a state trace. There are different XML structures possible. You can describe all components separately and than describe all connection with a list of links. One possible structure is shown in Figure 3-12 (note: It is the structure scenario pictured in Figure 3-11). The measurement data can be stored into the tags of services or links.
3.10 XML Representation

Another possibility is to use the table structure of the database for storing the information into an XML file. The scenario shown in Figure 3-11 may look like the following:

```xml
<Server1>
  <Service1 />
</Server1>

<Server2>
  <Service2 />
</Server2>

<Server3>
  <Service3 />
</Server3>

<Links>
  <Link from="Service1" to="Service2" />
  <Link from="Service1" to="Service3" />
</Links>
```

**Figure 3-12: XML representation 1**

Of course there are more possibilities but the second representation is used in this work because it reflects the used database structure and this is useful when searching for particular measurements or the like.

3.10.1 Current State

The current state depends on the measurement period \((t, \Delta t)\), thus there is time information needed in the XML representation. This could be a simple tag, which contains the point of time, when the period ends and the duration of the period in seconds. If we use global time information we do not need to store the time with any value. That way storage space for the XML-file can be reduced. The next step is to store the measurements in the XML-file. This can
be realized by using the table structure described in chapter 3.10. Thus the resulting structure will be:

```xml
<Time value="2002-03-01 08:23:20" duration="300" />
<Server1>
  <Service1>
    <request>
      </request>
    <process>
      </process>
  </Service1>
  <Remote>
    <Service2>
      <request>
        </request>
      <process>
        </process>
    </Service2>
    ... 
  </Remote>
  </Service1>
</Server1>
... 
```

Figure 3-14: Current state XML representation

The values of the measurements can be stored into tag “<value>”. The resulting XML-file will have the following structure:

```xml
<Time value="2002-03-01 08:23:20" duration="300" />
<Server1>
  <Service1>
    <request>
      <Value>12</Value>
      </request>
    <process>
      <Value>0.412</Value>
      </process>
  </Service1>
  <Remote>
    <Service2>
      <request>
        <Value>3</Value>
        </request>
      <process>
        <Value>1.04</Value>
        </process>
    </Service2>
    ... 
  </Remote>
  </Service1>
</Server1>
... 
```

Figure 3-15: Current state
3.10.2 Trace

The structure of current state representation can be used for the trace representation. The difference is that there are more values of measurement and the duration property is not needed any more. A simple way to realize this is to use a sequence of "<Time>-tags by giving them a sequential number. The same should be done with the "<Value>-tag. Thus we have global time information about the trace and all values in one XML-file.

```xml
<Time number="1" value="2002-03-01 08:23:20" />
<Time number="2" value="2002-03-01 08:23:25" />
<Time number="3" value="2002-03-01 08:23:30" />

...<Server1>
  <Service1>
    <request>
      <Value number="1">12</Value>
      <Value number="2">2</Value>
      <Value number="3">4</Value>
    </request>
  </Service1>
</Server1>
...
```

Figure 3-16: Trace representation
4 Distributed Media Content

4.1 Overview

Media-on-demand has become popular, especially video-on-demand but also music-on-demand. There are different system structures possible, but here we use the following: A media-on-demand system has three components: Clients (Customer), a network and media archive servers. Different media archive server will be distributed worldwide to provide clients with media content. Media archive servers are classified to content provider and media center. Media center are minor media archives, which are placed in each housing area. Clients will send all requests directly to these media centers and are directly connected to them. If the content a client asks for is not stored on the media center, the media center will ask a neighboring media center or a content provider. They function as a local cache for media content. Content providers are centralized servers, which distribute the media content to the media centers. The content on providers may vary between different regions.

![Figure 4-1: Distributed media content scenario](image)

4.2 System Model

To describe the scenario there are two categories of parameters: System characteristics and customer behavior [LTRA], [TFIC], [IMA].

4.2.1 System Characteristics

Characteristic parameters, which describe the system capacity, are for example:

- Number of simultaneous media streams a media server can support
- Storage capacity of a media server
- Available media content
- System structure
- Network bandwidth of the connections between separate media servers
- Number of customers connected to a media server
- Policy of getting media content
- Processing time of a request

4.2.2 Customer Behavior

For customer we can define the following parameters in order to describe their behavior:

- The request arrival pattern or workload at the server
- Media content selection patterns

4.2.3 System Structure

The distributed media content system is structured hierarchically. The upper hierarchy consists of the content provider, the next hierarchy consists of the media center and the bottom hierarchy consists of the customers. Customers are connected to one media center and media centers are connected to several content providers and also to media centers in their neighborhood. The system structure is shown in Figure 4-2. We can assume that about 100 to 1000 customers are connected to a media center and basically every media center to every content provider. Media centers are also connected to media centers in their neighborhood to compensate failures on single media centers by using the neighboring media center's services.
4.2 System Model

4.2.4 Typical Parameters of Media Contents

If we assume that movies are compressed with the MPEG-1 technology [MPG] a movie transmission will have a rate of 1.5 Mbps [OVD]. With a typical length of about 105 to 135 minutes a movie needs between 1.1 GB and 1.4 GB of storage capacity. Music data without compression has a rate of 1.4 Mbps (44.1 kHz, 2 channels with 16 Bit), by using compression like MP3 or similar there are rates of 128 kbps possible. Thus a whole CD of audio data needs approximately 60 MB of storage capacity.

4.2.5 Typical Customer Behavior

Customers have a typical request pattern by using media content: There is a “prime-time” during the day, which is characterized by high workload on the server because several customers want to see a movie. The rest of the day customers may only listen to music or the workload is low because people work or sleep. This “prime time” depends on the region of country (e.g. Germany: 9:00 p.m. or Spain: 10:30 p.m. [TSOV]).

Finally we can classify three phases of customer behavior: Sleeping time, prime time and the remaining time. At the prime time everybody wants to see a movie, and so there are high rates of movie requests at the media centers. The sleeping time is a period when nobody wants to see a movie or listen to any music. The request rate on media centers is very low at this time. At the remaining time people only listen to music and sometimes watch a movie. The simplified behavior of a customer is shown in Figure 4-3.
Customers do not choose the content completely randomly. There are some movies or music contents customers often want to see or listen to it. These are current contents like new movies or new songs.

4.3 Service Models

The goal of monitoring a distributed media content environment and its service model is to get information about the utilization of special media server, the relations between different media centers and the used network capacities. The model described in chapter 3 delivers all these information, thus we can completely use this model for the monitoring of distributed media content environments. Services are the particular media server (media center and content provider) and customers are creating requests to media servers.

4.3.1 Sensor

The sensors are measuring the parameters of distributed services as described in 3.8.2. Exceptions are the parameter \( u_s(\Delta t) \) and \( w_{n,m}(\Delta t) \). Normally they would represent the time in which a service completes its work; here this would nearly (processing delays) be the movie length or the music duration because the service is complete if the stream is transmitted. More important for this scenario is to measure the processing time, which is needed before a stream begins. So \( u_s(\Delta t) \) is the summated time between the incoming request and the start of transmission. With \( w_{n,m}(\Delta t) \) it is similar. After the measuring period the sensor sends the parameter to the database. The database structure is described in 4.3.3. The adjusted sensor notation is shown in Figure 4-4. Basically, it is the same as in Figure 3-8, but for better understanding the process states are renamed.
4.3 Service Models

4.3.2 Summary

For the service model of the distributed media content environment we can summarize:

System Configuration

\[ M : \quad \text{Available media content divided into music content and movie content} \]
\[ \quad \text{a number represents a media title.} \]

\[ \frac{1}{\mu_n} : \quad \text{Request processing time of media center } n \text{ (time between incoming request and} \]
\[ \quad \text{start of transmission)} \]

\[ m_n : \quad \text{Storage capacity of media center } n \]

\[ c_n : \quad \text{Number of customers connected to media center } n \]

\[ \bar{\eta}_{n,m} : \quad \text{Network capacity in KB from media center } n \text{ to content provider } m \]

\[ M_m \subseteq M : \quad \text{Media content on content provider } m \]

\[ \frac{1}{\mu_m} : \quad \text{Request processing time of content provider } m \]

\[ \chi_m : \quad \text{Available transmission channels on content provider } m \]
Model Parameter

In comparison to chapter 3.5 we can define the following parameters, which reflect the current state of the model:

Media Center

\[ \lambda_{0,n}^1(t, \Delta t) : \text{Rate of customer request rate to media center } n \]

\[ \mu_{n}^1(t, \Delta t) : \text{Request processing capacity of media center } n \]

\[ \frac{1}{\mu_{n}^1(t, \Delta t)} : \text{Mean time to process one request at media center } n \]

\[ \rho_{n}^1(t, \Delta t) = \frac{\lambda_{0,n}^1(t, \Delta t)}{\mu_{n}^1(t, \Delta t)} : \text{Utilization of media center } n \]

\[ \omega_{0,n}^1(t, \Delta t) : \text{Mean time one customer request takes to media center } n \]

\[ \eta_{n,m}^1(t, \Delta t) : \text{Network load in KB from media center } n \text{ to content provider } m \]

Content Provider

\[ \lambda_{n,m}^2(t, \Delta t) : \text{Request rate from media center } n \text{ to content provider } m \]

\[ \mu_{m}^2(t, \Delta t) : \text{Request processing capacity of content provider } m \]

\[ \frac{1}{\mu_{m}^2(t, \Delta t)} : \text{Mean time to process one request at content provider } m \]

\[ \rho_{m}^2(t, \Delta t) = \frac{\sum_{n} \lambda_{n,m}^2(t, \Delta t)}{\mu_{m}^2(t, \Delta t)} : \text{Utilization of content provider } m \]

\[ \omega_{n,m}^2(t, \Delta t) : \text{Mean time one request takes from media center } n \text{ to content provider } m \]

Measures

The following measures are needed to get the model parameter:

\[ \Delta t : \text{Period of time the measurement runs} \]
4.3 Service Models

**Media Center**

\[ r_n^1(\Delta t) : \text{Number of requests to media center } n \text{ in period } \Delta t \]

\[ u_n^1(\Delta t) : \text{Customer request processing time on media center } n \text{ in period } \Delta t \]

\[ r_{n,m}(\Delta t) : \text{Number of requests from media center } n \text{ to content provider } m \text{ in period } \Delta t \]

\[ w_{n,m}(\Delta t) : \text{Time the media center } n \text{ waits for content provider } m \text{ in period } \Delta t \]

\[ n_{n,m}(\Delta t) : \text{Summated network load of requests from media center } n \text{ to content provider } m \text{ in period } \Delta t \]

**Content Provider**

\[ r_m^2(\Delta t) : \text{Number of requests to content provider } m \text{ in period } \Delta t \]

\[ u_m^2(\Delta t) : \text{Media center request-processing time on content provider } m \text{ in period } \Delta t \]

### 4.3.3 Database Storage

The database storage is realized as described in 3.9, thus the resulting table structure is:

```
Content Providers:          Media Centers:
DistributedMediaContent    +MediaCenter1Host
  +ContentProvider1Host    +  MediaCenter1
      +ContentProvider1
          +request
          +process
      +ContentProvider2Host
          +ContentProvider2
              +request
              +process
      +ContentProvider3Host
          +ContentProvider3
              +request
              +process
...                        ...
```

*Figure 4-5: Table structure in database*
4.3.4 XML Representation

The XML-representation of the distributed media content environment’s state is realized as described in chapter 3.10. The only differences are the names of the servers and services.

Current State

```xml
<Time value="..." duration="...">
    <MediaCenter1Host>
        <MediaCenter1>
            <request>
                <Value>...</Value>
            </request>
            <process>
                <Value>...</Value>
            </process>
            <Remote>
                <ContentProvider2>
                    <request>
                        <Value>...</Value>
                    </request>
                    <process>
                        <Value>...</Value>
                    </process>
                </ContentProvider2>
            </Remote>
            <Remote>
                <ContentProvider1>
                    <request>
                        <Value number="1">...</Value>
                        <Value number="2">...</Value>
                        <Value number="3">...</Value>
                    </request>
                </ContentProvider1>
            </Remote>
        </MediaCenter1>
    </MediaCenter1Host>
```

Figure 4-6: Current state

Trace

```xml
<Time number="1" value="...">
    <Time number="2" value="...">
        <Time number="3" value="...">
            <ContentProvider1Host>
                <ContentProvider1>
                    <request>
                        <Value number="1">...</Value>
                        <Value number="2">...</Value>
                        <Value number="3">...</Value>
                    </request>
                </ContentProvider1>
            </ContentProvider1Host>
        </Time>
    </Time>
</Time>
```

Figure 4-7: Trace representation
5 Simulation with Service Models

5.1 Introduction

The goal of the simulation is to store monitor measurements in a database that are similar to the measurements of real systems. Thus you can identify the typical customer behavior (request pattern) and the media transmission through the network infrastructure.

5.1.1 Model Structure

The simulation model includes all components of the distributed media content scenario: Customers are connected to a media center and the media centers are connected to different content providers. The resulting structure is as shown in Figure 4-2.

5.1.2 Policy

The policy gives the working strategy of the particular system components. It is important that we are not concerned with any kind of failures, such as a down content provider or media center. We assume that all components are working during the simulation.

Content Provider

A content provider is waiting for a request from a media center. If a media center requests media content, the content provider reserves a transmission channel for this transmission. If the content provider has no channel available, the request is refused. To see how many requests are refused we add a counting variable to the sensor, which is used like the sensor’s request counter. After a transmission the content provider de-allocates the used resources.

Media Center

Media centers are waiting for requests from a customer. After a customer’s request for a media title the media center searches the content in its local cache. If it is available, the media center streams the content continuously to the customer. If it is not available, the media center asks the content providers, which are connected to the media center, for this content. When a content provider starts the transmission of the content to the media center, the stream will be handed over to the customer and stored in the media center cache synchronously. For media replacement in the cache an LRU-algorithm is used. The connection to a neighboring media center will not be treated with.
Customer

Customers are the producer of workload in the simulation. They are requesting media content as described in chapter 4.2.2. It is important that customers do not use any VCR-functions such as fast-forward or stop and replay. This simplifies the simulation so that a customer only requests media content one after the other.

5.1.3 Requirements

It should be simple to configure the simulation. That means to control the customer’s behavior and which components (e.g. customer) are connected to which other component (e.g. media server). And it is important that the simulation is independent from the computing power of the hosts, which process the simulation. Therefore we need a virtual time for simulation.

5.2 Methods

In practice there are various simulation methods [MSIM] for different tasks. The simulation system consists typically of various interacting objects. Each object changes its state in time. The different simulation methods differ in way of the simulation time goes by:

Continuous Simulation

The simulation time goes by continuously. This is useful for simulating models of differential equations.

Time-Based Simulation

The simulation time goes by in discrete steps $\Delta$. Some points of time perform events, which were processed. The disadvantage is that there are points of time at which no event (death time) is performed, but they were considered too. That means a waste of computing time.

Event-Based Simulation

The simulating time continues to the respective next event. That is a great advantage because we are not concerned with points of time the system state is not changing. Thus we have no death time.

For this simulation we use the event-based simulation, because it is a method, which computes only the important events. This simulation method will be the fastest.
5.3 Synchronization

There are different objects in the environment of distributed media contents (content provider, media center and customer). Thus it is possible to separate them in the simulation as different processes. The distribution has the positive effect of speedup if all objects are independent as much as possible so that the synchronization of event processing is as idle as possible.

For synchronization we use a conservative simulation algorithm because of the easier handling and implementation. In this simulation algorithm every simulation process handles its own event list until there is an event, which may depend on events of other processes. This form of dependence is considered to be causal. Two events are causal (noted: $e_1 \rightarrow e_2$) if:

- The time of $e_1$ is before the time of $e_2$ and:
  - Event $e_1$ changes a state variable that event $e_2$ reads
  - Event $e_1$ reads a state variable that event $e_2$ changes
  - Event $e_1$ changes a state variable that event $e_2$ changes too

- There is an event $e_x$ with $e_1 \rightarrow e_x$ and $e_x \rightarrow e_2$ (transitive)

A simple example is the simulation of two intersections with two processes. A car drives out of intersection 1 into intersection 2. The event "incoming car" on intersection 2 must be after the event "outgoing car" on intersection 1.
6 Implementation

6.1 Simulation

The simulation is implemented with the help of the programming language C, runs on Linux systems and uses the Message Passing Interface [MPI] for distributed computing. Thus the simulation system is able to run on the CLIC. For parsing the XML configuration-file the free XML-library “libxml” [LIBX] is used.

6.1.1 Processes

The simulation is divided into 2 process types. The first is the process, which contains the functionality of all content providers. The second contains the functionality of a media center with its connected customers. For a scenario of 5 content providers and 10 media centers we need 11 processes (one for the 5 content providers, ten for the 10 media centers). The processes use the MPI for inter-process communication (IPC). The resulting scenario is shown in Figure 6-1.

General Behavior

Each process (content provider and media center) has its own event list. An event list is a time-ordered list of events. Time means the point of time the event happens. For virtual time we use a resolution of one second. Important is that every event needs no time for happening. An event may contain special information, which depends on the event type.
A process takes the first event (next point of time of the simulation) from the list and handles this event. The following options are possible: Creating new events or changing a process state variable. If the event is processed it will be removed from the list.

The initial event lists of every process were generated at the start of the processes (simulation start). An initial event, which is generated on every process, is the event “Simulation-End”. If a process takes this event from the list, it will finish its work. So we have a controlled behavior for ending the simulation.

**Content Provider**

All content providers were simulated in one process. Thus the media center processes communicate to one process only. This reduces the inter-process time-synchronization.

A content provider has the following status information:

- Movie contents stored on the content provider
- Music contents stored on the content provider
- Number of free transmission channels

Globally for all content providers, there is an array that is used to store the current time of all media centers. This is used to avoid the violation of causal events. The content provider handles only events whose point of time is earlier or equal to the minimum time of all media centers. So the content provider process handles no event until there is no causal event at each media centers process any more

The content provider process communicates with media center processes through MPI-messages. These messages contain a time stamp, which presents the point of time the message was send. Thus the content provider process can update the current time array of the media centers. There are three types of messages:

- Media requests: After receiving this message, a “Media Request” event is inserted into the list.
- Time synchronization: This message is only used for updating the current time array.
- Reply: After a media request, this message is send back to the media center process.
6.1 Simulation

The content provider process knows the following events:

- **“Simulation-End”**: This event is used for a controlled program termination.

- **“Media Request”**: This event is used to handle the media request of a media center to a content provider. If the content provider has stored the requested media content and a free channel is available, the content provider reserves a channel and inserts a “Transmission End”-event into the list. At last the content provider process sends a message to the media center process, which has requested the content, to indicate the success of the request.

- **“Transmission End”**: This event simulates the end of a media transmission; the used channel is de-allocated after this event.

- **“Sensor Transmission”**: This cyclic event is used to perform the transmission of the data to the database, which was collected by the sensor. Every content provider has its own sensor data, thus by handling this event the data of all content providers is sent.

The policy of processing the content providers is:

1. Get the next event from the event-list

2. If the point of time of this event is after the minimum time of all media center step to 4

3. Process the event and go to step 1.

4. Wait for messages from a media center.

5. Process the message and go to step 1.

**Media Center**

A single process represents every media center. Media center processes simulate the behavior of the media center and the behavior of connected customers, too. A media center has the following status information:

- Cached movie contents stored on the media center

- Cached music contents stored on the media center
Free network resources to each content provider

The MPI-messages are the same as the messages from the content provider because the messages are transmitted between these components.

The following events are used in media center processes:

- **"Simulation-End"**: This event is also used for a controlled program termination.

- **"Media Request"**: This event is used for simulating a customer request to the media center. The media center searches for the requested media content in its local cache. If the content is not found, the media center will send a "Media Request"-message to the content providers, whose connection has free network capacity, and wait for a reply-message. The cached media content will be replaced (in case the cache is full) by using the LRU algorithm. Once the content is available a "Transmission Begin"-event is inserted into the event-list and the network capacity for transmission from the content provider is reserved (if in use). If the content is not available, a new "Media Request"-event is created.

- **"Transmission Begin"**: This event simulates the start of a media transmission to a customer. With this event a "Transmission End"-event or a "Transmission"-event is created, depending on the source of the content (content can be stored local or be provided by the content provider).

- **"Transmission"**: This event simulates the network traffic between a media center and a content provider, by adding the number of transmitted bytes to the sensors data. This event will occur every second until the content is transmitted. After that a "Transmission End"-event will be created.

- **"Transmission End"**: This event simulates the completion of a media transmission. Thus the used network capacity will be de-allocated. Following this, a new "Media Request"-event will be inserted into the event-list to simulate the next request of this customer.

- **"Time Synchronization"**: This event is used for updating the current time of the media center on the array of the content provider process. It is created cyclical to guarantee a continuous synchronization.
“Sensor Transmission”: This cyclic event is used to perform the data transmission from the sensor to the database.

The policy of a media center process consists only of processing the event-list. If a media center sends a media request to a content provider, the media center waits for a reply. This reply is only sent by the content provider if all media centers have the same current time or later. Therefore we have a global synchronization through the content provider process.

Media requests from customers are depending on particular daytimes (see 4.2.5). Every daytime period has its own probability for the choice between movie and music and a special probabilistic delay between two customer requests.

Customers of a media center have their own preferred “top-movies” and their own preferred “top-music”, the media content, which is liked requested (see 4.2.5). This is realized by a list of “top-medias” and the probability of choosing one of them.

Example

For a better understanding of the behavior of the processes, we will provide a simple example of one content provider and one media center. There is one customer who requests a particular movie (at time = 10). The media center’s cache is empty and has to send a request to the content provider. After that the simulation will end (at time = 10000). To simplify this example we are not deal with sensor specific events “Transmission” or the like:

Step 1: The initial event lists are created.

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Media Request</td>
</tr>
<tr>
<td>10000</td>
<td>Simulation-End</td>
</tr>
</tbody>
</table>
```

Step 2: The content provider waits for messages from the media center since the “Simulation-End”-event is causal to the “Media Request”-event from the media center. The media center process handles the first event (“Media Request”) and sends a request-message to the content provider and waits for a reply.

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Media Request</td>
</tr>
<tr>
<td>10000</td>
<td>Simulation-End</td>
</tr>
</tbody>
</table>
```
Step 3: The content provider receives the message and inserts a “Media Request”-event into its list (with the current time of the message). This event can be processed because the media center is at the same current time as the event. The content provider has the media content and reserves a channel for transmission and sends a reply-message to the media center.

Step 4: The “Media Request”-event is deleted from the content provider’s list; network capacity is reserved for transmission and a new event “Transmission End” is added to the list (the movie needs 36 seconds). After that the content provider has to wait for messages from the media center (causality). The media center removes the “Media Request”-event and inserts a “Transmission End”-event, too.

Step 5: The media center handles the “Transmission End”-event (de-allocate used network capacity) and removes it from the list.
6.1 Simulation

Step 6: The media center processes the “Simulation-End”-event and sends a time synchronization message to the content provider. Thus the content provider can handle the “Transmission End”-event (de-allocates used channel) and the “Simulation-End”-event. After that the simulation is finished.

Content provider: 46 Trans. End 10000 Simulation-End

Media center: 10000 Simulation-End

Content provider: 46 Trans. End 10000 Simulation-End

Media center: finished

Content provider: 10000 Simulation-End

Media center: finished

Figure 6-7: Example (final steps)

6.1.2 Configuration

The simulation should be configurable in order to run different simulation-configurations with the same simulation system. The simulation can be configured through an XML-file, which contains the information about the system structure and the customer’s behavior.

Global Information

At first there are several global information, which are valid for all processes:

- Time when simulation starts: Represented by tag “<Start>”, contains the full date and time of the point of time the simulation starts (e.g.: 2002-03-01 08:00:00).

- Time when simulation ends: Represented by tag “<End>”, contains the full date and the point of time the simulation ends (see above).

- Length of measurement period of the sensors: Represented by tag “<SensorPeriod>”, contains the number of seconds a sensor measurement period takes.
Name of the database server: Represented by tag “<DatabaseServer>”, contains the hostname of the database server.

Folder in database tables to store measurements: Represented by tag “<DatabaseFolder>”, contains the name of the folder, which is used for storing the measurements.

Available media contents: All available media contents are collected under the tag “<Medias>”, separated in movie content (“<Movie>”) and music content (“<Music>”). Media content is represented by tag “<Media>” and contains the “duration”, “data rate” and a continuous “number” for identification. Duration gives the length of media content in seconds and the data rate is given in Kbps. Because of the possibility that there are more than 1000 available media contents we need the functionality of generating media content. This functionality is also implemented: With the tag “<Generate>” we can generate media contents, which have the same data-rate specified by the property “datarate” and a specific duration configurable by the properties “minduration” and “maxduration”. The number of the generated media contents can be configured with the help of the property “count”.

<Start>2002-03-10 00:00:00</Start>
<End>2002-07-10 23:59:59</End>
<SensorPeriod>60</SensorPeriod>
<DatabaseServer>domitian.informatik.tu-chemnitz.de</DatabaseServer>
<DatabaseFolder>CLICSim</DatabaseFolder>
<Medias>
  <Movie>
    <Media number="1" duration="7380" datarate="1500" />
    <Generate count="2000" minduration="7000" maxduration="8000" datarate="1500" />
  ...
  </Movie>
  <Music>
    <Media number="1" duration="320" datarate="128" />
    <Generate count="2000" minduration="180" maxduration="300" datarate="128" />
  ...
  </Music>
</Medias>

Figure 6-8: Example of global configuration
Content Provider

All content providers are collected by tag “<ContentProviders>” and separated by tag “<ContentProvider>”. Every content provider has a continuous identification number, which is represented by the property “number”. For the content provider we need the following information:

- The stored media content on the content provider: The media content, stored on the content provider is represented like the global media content. But we do not need the duration and the data rate, because those are global parameters. Thus we only need the number, which refers to the number of the global media content. It is also possible to generate a set of stored media contents by using tag “<Generate>”. The tag has the properties “begin” and “end” which means the first number and the last number of the media contents that are generated.

- The number of available transmission channels: Represented by tag “<Channels>”.

- The processing delay for a request: Represented by tag “<Delay>”, contains the processing time in seconds (floating point values are possible).

Have a look at this short example:

```xml
<ContentProviders>
  <ContentProvider number="1">
    <Channels>100</Channels>
    <Delay>0.123</Delay>
    <Medias>
      <Movie>
        <Media number="5" />
        <Media number="8" />
        ...
      </Movie>
      <Music>
        <Generate begin="12" end="104" />
        ...
      </Music>
    </Medias>
  </ContentProvider>
  ...
</ContentProviders>
```

Figure 6-9: Example of content provider configuration
Media Center

All media centers are collected by the tag “<MediaCenters>” and separated by the tag “<MediaCenter>”. Every media center has a continuous identification number, which is represented by the property “number”.

For the media center we need the following configuration:

- Size of the local cache: Represented by the tag “<Capacity>”, contains the number of media contents, which can be stored on the media center, divided into music content and movie content (properties: “movie”, “music”).

- Processing delay for a request: Represented by the tag “<Delay>” similar to the delay from a content provider.

- Connected content provider and the network bandwidth of the link: Collected by the tag “<Connected>”, contains the content provider (tag “<ContentProvider>” with property “number”, which refers to the content provider identification number) and the bandwidth of the connection (property “network”) in Kbps.

- Number of connected customers: Represented by the tag “<Customer>”.

- Preload media content: Similar to media content on a content provider but without generating functionality.

- “Top-movies” and “Top-music” of the customers: Collected by the tag “<TopMedias>”, which contains the probability of choosing a top media in percent represented by the property “probability”. The media contents are represented as the stored medias on a content provider.

- Time periods of customer behavior: Collected by the tag “<Periods>”. A single period is represented by the tag “<Period>”, which contains the beginning of the period (property “begin”), the end of the period (property “end”), the movie-selection-rate in percent (property “movieprob”) and the mean time between a customer requests two media titles (property “mtime”).

A simple example is shown in Figure 6-10.
Sequence

The simulation sequence starts with the distribution of processes, just as normal MPI-applications. The next step is that every process reads the configuration from the XML-file. After the configuration, it is possible that the configuration differs from process to process because of the random generation of media content (see 6.1.2). Transmitting all media contents from process 0 to all other media center processes, through MPI-messages compensates this. Then the process with the MPI-identification number 0 creates all needed tables in the database for storing monitor data.
After that every process becomes its own task: One process simulates the content provider, the other processes will simulate media centers with connected customers. Process 0 simulates all content providers.

Now every process creates its initial event list. The content provider only inserts the “Simulation-End”-event and the first “Sensor Transmission”-event into its list. Media centers additionally insert the next customer requests of all customers and the first “Time synchronization”-event to their event-list.

6.2 Browser Adapter

The browser adapter should get the data stored in database, not only the data of one service, but the data of the whole system. What the representation looks like is described in 3.10.
6.2 Browser Adapter

6.2.1 Overview

For getting the model measurements we must upgrade the browser adapter of the precursory work. We need to modify the XML-Interface and the Browser-Adapter module shown in Figure 6-13. It is also possible to only modify the Browser-Adapter module, but we use this way because it is the more efficient solution (slow XML-messaging).

The Browser-Adapter module needs an extension to query a model measurement series. Moreover it needs an extension to generate an XML-message for the XML-Interface module to get all needed data. The XML-Interface module needs an extension to process these messages and should deliver the measurements through an XML-file.

![Figure 6-13: Modification of the Browser Adapter](image)

6.2.2 XML-Message

The message for getting the model measurement series is similar to ordinary query messages. A substantial difference is that the table-select path is not a value table, but the root-folder (see 3.9) of the model measurements. The interval of the query also contains an entry for the sensor-measuring period.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<body>
  <modelQuery>
    <tableselect>
      <sub name="root" />
      <sub name="CLICSim" />
    </tableselect>
    <interval>
      <begin dateTime="2002-01-23 15:00:00" />
      <end dateTime="2002-01-23 15:30:00" />
      <period second="120" />
    </interval>
  </modelQuery>
</body>
```

![Figure 6-14: XML-message for model measurements query](image)
7 Examples

In this chapter we investigate two example scenarios in order to demonstrate how service models and measurements looks like and which results can be derived. Measurements are generated by the simulation system.

7.1 A Cause for Media Centers

The question considered here is: Is there an advantage to use media centers as a local cache for distributed media content? To analyze this question we compare two scenarios: One with media centers and the other without media centers with customers directly connected to the content providers. To simulate direct connected customers (to content provider) we assume the local storage and the processing time of a media center to be zero.

![Diagram of two scenarios for simulation](image)

**Figure 7-1: Two scenarios for simulation**

7.1.1 Service Model Configuration

In this example we use a scenario of 3 content providers and 10 media centers with 2000 available media titles (1000 music and 1000 movie), which are distributed across the 3 content providers. Each content provider stores 600 titles (300 music and 300 movie) exclusively and
200 medias (100 music and 100 movie) are available on all content providers, these may are news or something else. Movie content uses a data rate of 1,500 Mbps and music content uses a data rate of 128 Kbps. The duration of a movie is between 1,5 hours and 2 hours. The 100 movies stored on all content providers have duration times between 15 and 30 minutes. The duration of music content is between 3 and 5 minutes.

Each media center is connected to every content provider through a 100Mbit link and has 100 customers online. There are two groups of customers: The first group accesses content in a prime time between 08:00 p.m. and 10:30 p.m., the second in a prime time between 05:00 p.m. and 07:30 p.m. Each group consists of 5 media centers. These two groups simulates customers in two time zones:

<table>
<thead>
<tr>
<th>Remaining time</th>
<th>Prime time</th>
</tr>
</thead>
<tbody>
<tr>
<td>05:00</td>
<td>05:00 p.m. 07:30 p.m.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remaining time</th>
<th>Prime time</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00</td>
<td>08:00 p.m. 10:30 p.m.</td>
</tr>
</tbody>
</table>

The top medias are equal for each customer: One movie from every content provider and three movies, which are available everywhere (stored on each content provider). A content provider has 100 transmission channels and a processing time of 0,5 seconds per request. A media center has a processing time of 0,25 seconds per request. These values are invented but sufficient for this simulation.
We can summarize the values for the service model:

\[ M = \{\{1, 1000\}_{\text{movie}}, \{1, 1000\}_{\text{music}}\} \): 1000 movies and 1000 music contents. (Movie and music titles with number between 901 and 1000 are stored on every content provider)

\[ \frac{1}{\mu_n} = 0.25 \text{ sec} \]

\[ c_n = 100 \text{ Customers connected to a media center} \]

\[ \eta_{n,m} = 100 \text{ Mbits} \]

\[ M_1 = \{\{1, 300, 901, 1000\}_{\text{movie}}, \{1, 300, 901, 1000\}_{\text{music}}\} \): 300 movie titles (number from 1 to 300) exclusively stored and 200 movie titles (number from 901 to 1000) stored on all content providers. With music titles it is the same.

\[ M_2 = \{\{301, 600, 901, 1000\}_{\text{movie}}, \{301, 600, 901, 1000\}_{\text{music}}\} \): 300 movie titles (number from 301 to 600) exclusively stored and 200 movie titles (number from 901 to 1000) stored on all content providers. With music titles it is the same.

\[ M_3 = \{\{601, 1000\}_{\text{movie}}, \{601, 1000\}_{\text{music}}\} \): 300 movie titles (number from 601 to 300) exclusively stored and 200 movie titles (number from 901 to 1000) stored on all content providers. With music titles it is the same.

\[ \frac{1}{\mu_m} = 0.5 \text{ sec} \]

\[ \chi_m = 100 \text{ Transmission channels of a content provider} \]

The two scenarios differ in one parameter: The storage capacity of media centers.

\[ m_{n,\text{movie}} = 0; m_{n,\text{music}} = 0 \): Storage capacity of media center \( n \) in scenario 1

\[ m_{n,\text{movie}} = 120; m_{n,\text{music}} = 120 \): Storage capacity of media center \( n \) in scenario 2

For simulation we use time between 00:00 a.m. and 11:59 p.m. We simulate a complete day with all time periods of customer behavior. We use a measuring interval of \( \Delta t = 2 \text{ min} \).
7.1.2 Results

Both simulations for either scenario have been run three times and were compared against each other. Since simulation results were nearly the same we use the results of the first simulation run for analysis.

Figure 7-4 and Figure 7-5 show the request trace on content provider 1 \( (\sum_{n} x_{n,1}(t,60\text{sec}); n \in N, 1 \leq n \leq 10) \). The dashed line shows the rate of refused requests:

![Figure 7-4: Request trace on a content provider without media center](image)

![Figure 7-5: Request trace on a content provider with media center](image)
In Figure 7-4 two overload situations occur between 05:00 p.m. and 00:00 a.m. The request rate increases up to 60 requests per minute. For the content provider it is impossible to perform these requests because the content provider has only 100 transmission channels and a channel is used for duration of at least 3 minutes. The reason for this high request rate is that most of the requests are refused. In this case customers generate new requests after a short time and so there are so much request. In Figure 7-5 we see a trace with using media centers as local caches. Overload conditions have disappeared: The request rate is flat. This demonstrates the advantage of using media centers as local caches.

In Figure 7-6 we see an example of a media center request trace with a small peak at 08:15 p.m. (prime time). In the remaining time the request rate is constantly flat. After 09:00 p.m. the request rate decreases since customers are watching movies making no further requests.

Figure 7-6 shows the request trace on media center 1 ($\lambda^1_{0,1}(t,60 \text{ sec})$):
Figure 7-7 and Figure 7-8 show the network load between media center 1 and content provider 1 ($\eta_{1,1}(t,1\text{sec})$):

![Network-load without media centers](image1)

**Figure 7-7: Network-load without media centers**

![Network-load with media centers](image2)

**Figure 7-8: Network-load with media centers**

Figure 7-7 and Figure 7-8 show, that using media centers reduces network load.
### 7.2 Preloading Media Content

In the next scenario we examine the case of a new movie being released and launched at the same time nation-wide. The question is: Is it reasonable to preload the new movie to media centers before launch date.

#### 7.2.1 Service Model Configuration

In order to investigate this scenario, we look at a simple example of one content provider and one media center. The configuration parameters are alike the parameters of the first example, but now with a single content provider that provides the content:

\[
M = \{\{1, \ldots, 1000\}_{\text{movie}}, \{1, \ldots, 1000\}_{\text{music}}\}: 1000 \text{ movies and } 1000 \text{ music titles.}
\]

\[
\frac{1}{\mu_1} = 0.25 \text{ sec}
\]

\[c_1 = 100 \quad \text{Customers connected to media center}\]

\[
\eta_{1,1} = 100 \text{ Mbits}
\]

\[
M_1 = \{\{1, \ldots, 1000\}_{\text{movie}}, \{1, \ldots, 1000\}_{\text{music}}\}: \text{All media titles are stored on the content provider 1}
\]
\[
\frac{1}{\mu_1^2} = 0.5 \text{ sec}
\]

\[\chi_1 = 100 \text{ Transmission channels of content provider 1}\]

\[m_{1,\text{movie}} = 120; m_{1,\text{music}} = 120 \text{ Storage capacity of media center 1}\]

For simulation we also use time between 00:00 a.m. and 11:59 p.m. We use a measuring period of \(\Delta t = 2 \text{ min}\). All new movies are pre-loaded to the media center.

### 7.2.2 Results

Figure 7-10 and Figure 7-11 show that at this scenario preloading media content has no influence to the request trace at the content provider. Both traces are similar and constantly flat and so it is guaranteed that all customers can watch the released movie.

Figure 7-12 and Figure 7-13 show that preloading media content has no influence to network traffic \((\eta_{11}(t,1\text{ sec}))\) too. Through the day almost the same network traffic is produced.

![Figure 7-10: Request trace without preloading media content](image-url)
7.2 Preloading Media Content

Figure 7-11: Request trace with preloading media content

Figure 7-12: Network load without preloading media content
7.2.3 Conclusion

We have seen that preloading media content has no influence at this scenario. The strategy of caching media content on the media center may explain these results: After customers request a media title it will be stored on the local cache. If any customers want to watch the same movie, the connected media center request the media title from a content provider one times only (at the first request by a customer). The following request can be served from the local cache and there is no need to request a content provider again.
8 Conclusion

In this work we discussed a model of distributed service environments. This model describes the behavior of such environments, relations between its components and is related to queuing network models, which describes systems of communicating components. We showed which measurements are needed to get the model parameters, how to get these measurements and how to store them in the database developed in the precursory work. We also described a method to store model measurements in an XML-document.

The model is used to describe a distributed media content environment, and a simulation system that runs on the CLIC helps to generate measurements as in real systems. Two examples illustrate how the model can be used to analyze specific scenarios.

The simulation system is event-based and distributed. According to relevant literature, this simulation method uses approximately 30% of processor time for managing the event list. But the big advantage is to simulate important points in time only.

8.1 Possibilities for the Future

The simulation system is implemented to store only one data value in database by generating an XML-message. Experiments show that database and XML-messaging are bottlenecks of the simulation environment because every database access needs approximately one second. Thus simulating a whole day of customer behavior may take more than one hour of simulation time (depending on the configuration). Without storing values into the database it takes less than one minute. This effect may be reduced by caching measuring values in the sensor because storing more than one value in the database by using one XML-message only is faster than storing one by one.

The system model does not contain information about the hardware layer. You cannot describe the physical connection between (logical only) and the physical load on the components. We would need a possibility to combine application-level measurements with hardware-level measurements. For example: Three communicating service processes use the same physical network connection. Thus the connection capacity is shared.

The sensors assume that service processes are communicating synchronously. That means a service process waits for the completion of remote services, which the service process needs. This may deliver measuring failures by measuring asynchronous processes.
The connection between media centers is not been considered. Thus a next step to upgrade the simulation system is to implement this functionality.

Another future goal is the graphical representation of the system model. This may be a graph with parameters (e.g. network load or request rate). The cause is: Graphical representation better shows the state of the system than an XML-file.

8.2 Related Work

There are companies that practice content delivering already. One of them is Akamai [ACO]. This company’s platform (“EdgeSuite”) for content, streaming media and application deliveries comprises more than 13000 servers in more than 1000 networks in 63 countries. These servers deliver HTML documents, static images and streaming media for over 1300 content providers, including many of the most popular sites on the web. The used service hierarchy is similar to the hierarchy of distributed media content deliveries in this work: There are some servers, which deliver the content and there are some servers (so called “Edge”) that cache this content for customer-delivery:

That way content can be delivered by accessing the “Edges” only. This happens without using the whole Internet with its bottlenecks such as first mile bandwidth constrains and router/switch limitations.
Acronyms

CD          Compact Disc
CLIC        (German) Chemnitzer Linux Cluster
HTML        Hypertext Markup Language
HTTP        Hypertext Transport Protocol
IPC          Inter Process Communication
LAN          Local Area Network
LRU          Last Recently Used
MPEG        Motion Picture Experts Group
MPI          Message Passing Interface
SVG          Scalable Vector Graphics
VCR          Video Cassette Recorder
VoD          Video on Demand
XML          Extensible Markup Language
Figures

Figure 2-1: Simple structure in database................................................................. 11
Figure 2-2: Parameter table examples................................................................. 12
Figure 2-3: Message for data query................................................................. 12
Figure 2-4: Main components of the database.................................................. 13
Figure 3-1: Services on a server......................................................................... 15
Figure 3-2: The general states of a service process........................................... 16
Figure 3-3: States of a web-server service process............................................. 16
Figure 3-4: Period of time for the parameters.................................................... 16
Figure 3-5: A simple example of characteristic parameters............................. 17
Figure 3-6: States of distributed service processes.......................................... 17
Figure 3-7: Scenario for distributed services.................................................... 18
Figure 3-8: events logged by the sensor............................................................ 21
Figure 3-9: Information sequence of the sensors.............................................. 22
Figure 3-10: Possible database structure.......................................................... 24
Figure 3-11: Possible representation of the three-server system....................... 24
Figure 3-12: XML representation 1................................................................. 25
Figure 3-13: XML representation 2................................................................. 25
Figure 3-14: Current state XML representation.............................................. 26
Figure 3-15: Current state.................................................................................. 26
Figure 3-16: Trace representation...................................................................... 27
Figure 4-1: Distributed media content scenario.............................................. 29
Figure 4-2: System structure.............................................................................. 31
Figure 4-3: Simplified customer behavior....................................................... 32
Figure 4-4: Notations by the sensor................................................................. 33
Figure 4-5: Table structure in database............................................................ 35
Figure 4-6: Current state.................................................................................... 36
Figure 4-7: Trace representation....................................................................... 36
Figure 6-1: Processes of simulation................................................................. 41
Figure 6-2: Example (step 1)............................................................................ 45
Figure 6-3: Example (step 2)............................................................................ 46
Figure 6-4: Example (step 3)............................................................................ 46
Figure 6-5: Example (step 4)............................................................................ 46
Figure 6-6: Example (step 5)............................................................................ 46
Figures

Figure 6-7: Example (final steps).............................................................................................................. 47
Figure 6-8: Example of global configuration .......................................................................................... 48
Figure 6-9: Example of content provider configuration ....................................................................... 49
Figure 6-10: Example for media center configuration .......................................................................... 51
Figure 6-11: Sequence (step 1).................................................................................................................. 52
Figure 6-12: Sequence (step 2).................................................................................................................. 52
Figure 6-13: Modification of the Browser Adapter .................................................................................. 53
Figure 6-14: XML-message for model measurements query.................................................................... 53
Figure 7-1: Two scenarios for simulation ............................................................................................. 55
Figure 7-2: Media content distribution on content providers ............................................................. 56
Figure 7-3: Used time zones in the simulation ....................................................................................... 56
Figure 7-4: Request trace on a content provider without media center .............................................. 58
Figure 7-5: Request trace on a content provider with media center ................................................... 58
Figure 7-6: Request trace on media center 1 ......................................................................................... 59
Figure 7-7: Network-load without media centers ................................................................................ 60
Figure 7-8: Network-load with media centers ..................................................................................... 60
Figure 7-9: Total network transmissions produced by media center 1 .............................................. 61
Figure 7-10: Request trace without preloading media content ........................................................... 62
Figure 7-11: Request trace with preloading media content .................................................................. 63
Figure 7-12: Network load without preloading media content ............................................................ 63
Figure 7-13: Network load with preloading media content ................................................................. 64
Figure 8-1: Scenario of Akamai’s EdgeSuite ......................................................................................... 66
Bibliography

[ACO] Akamai Technologies Corporate
http://www.akamai.com/

[ALVS] Tran-Gia, P.
Analytische Leistungsbewertung verteilter Systeme
Springer., 1996

[AWS] The Apache HTTPD Project
http://httpd.apache.org/

[ICPE] Kant, K.
Introduction to Computer System Performance Evaluation

[IMA] Wittig, H.
Intelligent Media Agents
Vieweg, 1999

[LIBX] The XML C library for Gnome
http://xmlsoft.org/

[LTRA] Almeroth, K. C.
Long Term Resource Allocation in Video Delivery Systems
IBM Research Report (RC 20249), 1996

[MBS] Tanenbaum, A. S.
Moderne Betriebssysteme
Hanser, 1995

[MDS] Mansouri-Masani, M.
Monitoring of Distributed Systems
University of London, 1995

[MLR] Haas, M., Zorn, W.
Methodische Leistungsanalyse von Rechnersystemen
Oldenbourg, 1995

[MPI] The Message Passing Interface
http://www-unix.mcs.anl.gov/

[MPG] MPEG.ORG
http://www.mpeg.org/

[MSIM] Mehl, H.
Methoden verteilter Simulation
Vieweg, 1994

[MVS] Kurschl, W.
Monitoring von verteilten Systemen
Johannes Kepler Universität, Linz, 1995
[OVD] Rautenberg, M.
Organisation eines Servers für Video on Demand
VDI Verlag, 1998

[PCCS] Haerkort, B.R.
Performance of Computer Communication Systems
John Wiley & Sons, Ltd, 1999

[TFIC] Klava, H., Furth, B.
Techniques for improving the capacity of video-on-demand systems
Columbia University

[TSOV] Tagesspiegel Online Dienste Verlag GmbH
http://www.tagesspiegel.de/

[XMDB] Koenig, R., Reimann, C.
XML-based Monitoring Database
Chemnitz University of Technology, 2001

/XML] Extensible Markup Language
http://www.w3.org/XML/
Statement of Autonomy

With my signature I affirm that this work was written by myself without any help of strangers. I have not used any sources or aids, which are not denoted. This work was not used with another testing office before.

Chemnitz, 05/31/2002

Carsten Reimann