PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 45, NO. 2, PP. 139–156 (2001)

DISCRETE SIMULATION OF DISTRIBUTED SYSTEMS – PERFORMANCE EVALUATION OF A NOTIFICATION CHANNEL FEDERATION

József HOSSZÚ

Department of Telecommunication and Telematics
Budapest University of Technology and Economics
1117 Budapest, Pázmány Péter sétány 1/D, Hungary
e-mail: hosszu@ttt-atm.ttt.bme.hu

Received: 3 Nov., 2000; Revised: 1 Dec. 2000

Abstract

This paper presents how discrete simulation can be used for performance evaluation of distributed systems. With this methodology it is not needed to implement the system itself, only a model of proper specification is required. Simulation models for distributed systems can be easily adopted from other models which are already used in network simulations with good results. The tool that supports our measurements is a powerful telecom simulation platform, a simulations development environment that supports object-oriented programming. The model used for demonstration represents a notification channel federation including an arbitrary number of event suppliers and event consumers connected to a scalable network. Performance is evaluated for various configurations, and results are presented.

Keywords: event notification, modelling, simulation, performance evaluation.

1. Introduction

This paper introduces a methodology suitable for performance assessments of CORBA-based distributed systems. With simulation, even large-scale configurations can be examined which may be based on any CORBA implementations, such as real-time and radio access object computing frameworks. Usually CORBA implementations and software components using CORBA are examined in point-to-point topology networks, according to the point-to-point nature of GIOP (General Inter-ORB Protocol). In this case only low and middle scale configurations can be examined. On the other hand, simulation makes it possible to deal with a wide range of configurations without their actual implementation or assembly. Only a proper simulation model and parameters from real circumstances and existing implementations are needed for the assessments.

Event notification is essential in network management. Many components in a distributed network management system need an event notification mechanism, and so an event notification framework is expected to facilitate the development of the components. CORBA (Common Object Request Broker Architecture [7]) provides an object-oriented platform to build such a framework, but its performance

is not always sufficient for monitoring large-scale networks. Higher performance is rather necessary when message filters are applied in network elements.

The paper is organized as follows: Section 2 outlines the CORBA reference model and the Notification Service; Section 3 summarizes the performance evaluation methods of distributed systems, requirements of testing, and the basics of discrete event driven simulation; Section 4 introduces the simulation model for a service of federated notification channels and the applied simulator platform; Section 5 presents the measurements and comments on the results; Section 6 discusses the extensions to the preliminary model, and points to important ORB functionality that can be improved using the simulation results; and Section 7 presents concluding remarks.

2. On CORBA-based Distributed Systems

2.1. Synopsis of CORBA

CORBA is a distributed object computing middleware standard being defined by the Object Management Group (OMG). CORBA is designed to support the development of flexible and reusable distributed services and applications by (1) separating interfaces from (potentially remote) object implementations and (2) automating many common network programming tasks, such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching.

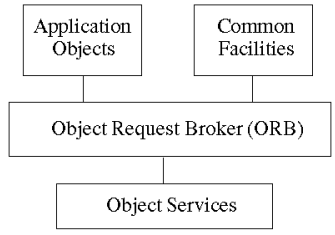


Fig. 1. OMG Reference Model Architecture

Fig. 1 illustrates the primary components in the OMG Reference Model Architecture [11]. At the heart of the OMG Reference Model is the Object Request Broker (ORB). CORBA ORBs allow clients to invoke operations on target object implementations without concern for where the object resides, what language the object is written in, the OS/hardware platform, or the type of communication protocols and networks used to interconnect distributed objects. The conceptual model developed by OMG is called the Object Management Architecture (OMA)[6] upon which applications can be constructed. The OMG OMA attempts to define, at a high level of abstraction, the various facilities necessary for distributed object-oriented

computing. The OMG OMA partitions the problem space into practical, high-level architectural components that can be addressed by technology proposers. It consists of four components: Object Request Broker (ORB), Object Services, Common Facilities, and Application Objects. These components define the composition of objects and their interfaces. Object Services is comprised of a set of interfaces to lower level infrastructure objects. The CORBAservices [5] specifications define sets of objects which perform fundamental functions such as naming, life cycle services, and transactions. Common Facilities are defined by OMA as having an application focus and mostly used by developers working in a distributed environment. Common Facilities provide standardized interfaces to common application interfaces. The Application Objects component reflects the need for independently developed application interfaces. While this is one of the components of the OMA, the OMG will probably never develop standards for it.

CORBA is an architecture for distributed objects. At the lower communication level, the object interaction takes the form of Remote Procedure Calls (RPCs), which are synchronous invocations. In addition, CORBA allows asynchronous interaction by specifying a oneway operation[1] semantics in which the client continues its computation without waiting for a result from the server after issuing the request. By using the Dynamic Invocation Interface (DII), CORBA supports deferred synchronous interactions in which the client is allowed to get a response from the server some time after issuing the request. Asynchronous communication is also supported by standard CORBA services. For details, see Section 2.2, and [5].

The Interface Definition Language (IDL) [7] is used to statically define the interfaces to objects, to allow invocation of operations on objects with differing underlying implementations. From the IDL definitions, it is possible to map CORBA objects into particular programming languages or object systems. An interface consists of a set of named operations and the parameters to these operations.

CORBA has designed a very basic inter-ORB protocol, called General Inter-ORB Protocol (GIOP), which serves as a common backbone protocol. The GIOP specifications are designed to be easily implementable on most transport systems. Internet Inter-ORB Protocol (IIOP) is one of the variants of the GIOP.

2.2. The Notification Service

Event notification is especially essential in network management systems. Network operators monitor and control the managed network through a suitable system. The network elements provide information to the operators on their state changes and failures. Using CORBA enables the development of a customizable and effective notification framework. Generally speaking, the servers of such a framework collect event information from the network elements and dispatch them to the client

[1]The oneway operation is a request-only operation with best-effort semantics, i.e. it does not return any result and the requester never synchronizes with the completion (if any) of the request.

components executed on the operator's hosts. The communication is based on asynchronous CORBA event messages. The proper standard CORBA service for this purpose is the CORBA Notification Service [8]. Notification Service extends the existing OMG Event Service [5], adding new capabilities to it (for details on OMG Event Service and Notification Service refer to [5] and [8], respectively). The OMG Event Service supports asynchronous exchange of event messages between clients. The Event Service introduces event channels which broker event messages, event suppliers which supply event messages, and event consumers which consume event messages.

The process of event delivery of the Notification Service can be summarized as follows. Two software components, the supplier that generates the event, and the consumer that receives it are using the Notification Service for their interactions and send asynchronous messages through the service to each other. The channel creation and connection management is done by the service itself. In a simple example the client and server objects are located on separated hosts (the separation is hidden by the transparency provided by the ORB). The hosts are connected to each other via a network connection. When issuing an event, it is first sent to the notification channel, and then the channel forwards it to the consumer. Hence, CORBA method invocations are mostly based on remote procedure calls, these steps cost at least two RPCs. There is a need, but no standard specification for interfaces that manage the creation and usage of networks of federated channels, because it raises many complex issues related to unique message identification, transactions and security [8]. If the supplier and the consumer cannot use the same notification service for their communication, or the service requires several connected (i.e. federated) channels, they may use several services and their channels at the same time. It is trivial that the cost of event delivery increases, and the number of RPCs is twice as many as the number of channels used. It also generates mathematical problems, e.g. problems with directed cycles in the network graph, too. There are certain CORBA implementations that include Event Service or Notification Service supporting some sort of federated event channels. For examples, see [3] and [9].

3. Methods for CORBA Performance Measurements

The primary characteristics of any distributed systems concerning time and duration as important factors are the following:

- Throughput: the amount of requests that the system is able to serve in a given time period.
- Latency: the time spent by the system, between issuing the request and receiving the response.
- Scalability: this parameter informs about the scale of the system, e.g. the possible number of processors or separated waiting queues, satisfying certain performance or QoS requirements.

- Overload possibility: this value assumes the rate at which the system becomes temporarily unusable or malfunctioning in certain circumstances.
- Concurrency: informs about the number of processes being executed in parallel.
- Distribution: determines the range of hosts and services used during the interactions in the system.

These characteristics form the metrics of the performance of distributed systems. Measurements are generally carried out with configurations of two hosts on a network (e.g. Ethernet) segment, and client objects on one of the hosts issue requests to the server objects on the other host. Requests are scheduled and the response times are measured. Additionally, compile time and executable program size may also be in the scope of examinations. Despite the large number of tests, no general considerations had been found which could serve as a basis of objective comparisons. Generally speaking, tests are performed in order to examine services, performance and interoperability of CORBA implementations.

The performance of a service is mostly measured with its throughput. One must also take into account the additional information available about the certain implementation. Performance of a notification server varies with factors coming into play. Filtering close to the supplier enhances performance as it reduces network traffic and supplier workload. Consumers are, in general, heavier CPU users than suppliers, which are more likely to be I/O bound. Busy channels are very sensitive to additional CPU-bound workload; raising channel priority may help in this case. These parameters mainly affect the communication of CORBA objects. As mentioned in Section 2.2, the communication is based on RPCs. For the evaluation or assessment of system efficiency, costs of the following steps have to be estimated: (1) parameter marshalling; (2) data transfer from the client to the server; (3) activation of the proper object implementation at the server; (4) demarshalling data at the server skeleton.

Several comparative tests have been carried out by researchers and developers. These are mostly benchmark tests based on remote method invocations and data transfer. [1] compared a Java based ORB with other CORBA implementations using various argument types and sizes with the invocations and round trip time results were presented.

[10] summarizes the experience of its author with distributed programming technologies under Linux. Different technologies and CORBA implementations are compared, and interoperability and performance were tested, but the methodology involved only execution of simple ‘Hello World’ applications.

[3] reports performance evaluation of the notification server bundled by release 5.1 of HP OpenView Communications CORBA Platform. They present experiment data and results of measurements for low-scale configurations (from 1 up to 3 hosts on private Ethernet), taking into account distribution, filtering and priority related issues. The main idea of the tests is invoking remote methods with various argument types and sizes.

A thorough, scientific investigation of CORBA performance is being performed by D. SCHMIDT. [9] presents their own real-time implementation of the

COS Event Service. The TAO Event Service is enhanced with federations support. However, it is an exhaustive evaluation, and TAO is developed for important and time critical purposes such as mission avionics. [9] does not report results or experience using the service for broadcasting on the Internet or other busy shared networks.

Another way of measuring (more accurately: assessing) the performance of a system is supplied by simulation. Several simulation methods exist, out of which discrete event driven simulation is considered to be a powerful, easy-to-implement paradigm. Discrete simulation maps the events and actions onto a virtual simulated time scale according to the simulation schedule and to the events that occur during the execution. When all events and actions are performed at the current simulated time, the virtual time pointer jumps directly to the next entry on the scale. Simulation makes it possible to evaluate the performance of complex systems, large-scale network configurations also considering the effects of real-life circumstances such as heavy traffic on the network, resource sharing, data loss, etc.

4. Simulator for Federated Channels

4.1. On the Simulations Platform

The PLASMA simulation development environment was developed by researchers and programmers at Ericsson Telecommunications Ltd., Hungary. The concept is the integration of network simulation with sophisticated mathematical algorithms, provides more efficient support for the analysis of network performance [2]. The simulation frame system has been also used for numerous other applications, e.g. an ATM network call level and cell level simulator, an intelligent network simulator and a mobile network simulator.

The simulator is based on a general discrete event driven simulation frame implemented using object-oriented methods. Objects that are common for every simulator are coded in PlasmaCORE. PLASMA simulator applications have the following main parts:

- Scheduler: Keeps track of the current value of simulated time as simulation proceeds.
- Agent: Controls the information flow between the simulated objects and the communication interface. For this purpose the simulator is extended by a Tcl interpreter.
- Measure Manager: The Measure Manager performs the creation of history from measurement attributes of the simulated objects.
- Simulated Objects: Simulated Objects are C++ modules that inherit communication, management and measurement abilities from a common ancestor. They represent a certain part of the simulation model and hence, they are simulation specific, simulated objects can perform different events. They are located in the Simulation Arena and can be connected to and communicate with each other.

In the simulator the modelled system is realised as a network of interconnected simulated objects, that communicate with each other entirely using pre-defined message forms. They may have an arbitrary number of attributes, which determine the state of them.

Results of simulated measurements can be read from the measurement attributes of simulated objects. Simulated measurement results are calculated on the basis of counting events during the simulation. The measurement period can be 1, 5, 15, 60 minutes of simulated time in accordance with ITU-T recommendations.

4.2. The Simulation Model

Because networks of federated channels can grow countrywide, e.g. an event notification framework of a national telecom operator, there is no chance to assemble the configuration and measure the overall performance prior to final installations. Simulation is useful instead, supported by a powerful telecom network simulation platform. This platform was Ericsson's PlasmaCORE in our measurements. An event notification network is simulated and its overall throughput is measured. The nodes of the network are elements that broadcast all incoming events. The edges of the network are notification channels. Filters can be assigned to each notification channel to prevent the network of flooding with unnecessary and multiplied instances of the events. Filters perform match operations on the type fields of the events, and accept only the types enlisted in the filtering constraints. Suppliers which generate, and Consumers which consume and destruct events are connected to the nodes of the network. Generally computers or hosts can behave both as suppliers and consumers; in this case such a host is modelled as a pair of one supplier and one consumer connected to the same network node.

The overall throughput is measured by means of calculating the time spent on forwarding all instances of the events. Latency measurements are carried out in order to determine the average time spent in the system by one event instance. Concurrency and scalability measurements are based on throughput measurements. These are supported by timestamps added to the events at generation and destruction, and the instance counters.

It is out of scope now, the problem includes but the simulation model lacks the powerful filter allocation algorithm which ensures that only the necessary events are forwarded through the channels. The importance of this algorithm is expressed, because the location of suppliers and consumers may differ dynamically as the network configuration changes. For simulation purposes a model for the notification channel federation was required. A network is built up of objects of the following types:

- Supplier: generates structured events of offered types. The inter-arrival times and length of the event sequences (structured events) are determined by customizable distributions.

- Consumer: consumes events, but accepts only those of the subscribed types. Sends measurement data gained from the timestamps carried by the event sequences to a global measurer object.
- Network Node: forwards the incoming events to the connected Notification Channels in a broadcast fashion, therefore unique message identification is necessary. A built-in delay stands for the time costs of remote procedure calls while delivering events to the channels.
- Notification Channel: the model for the event channel applied in the Notification Service, serving as connections between the nodes of the network. The internal structure of the Notification Channel model is depicted in *Fig.2*. The Notification Channel follows the push mechanism[2], which means that when an event arrives at its input, it is then scheduled to be forwarded to the output. A customizable data loss module stands for the possible loss of events on real connections (e.g. buffer overflow on TCP/IP networks, using IIOP as inter-ORB protocol). The built-in delay module represents the time costs of RPCs that occur during event delivery. Filter objects can be attached to the Notification Channel. To prevent the network from flooding with multiplied event instances, a powerful filter allocation algorithm is needed.

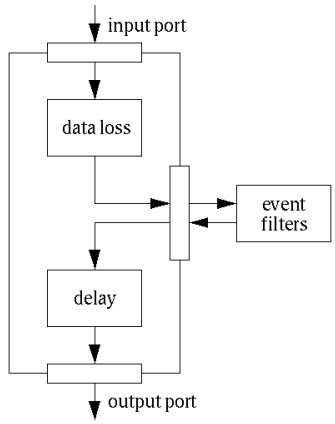


Fig. 2. Model of the Notification Channel

Measurements are based on the delivery of structured events (sequences of events), and the delivery time is measured. In our approach (as we are measuring the throughput of federated services) the delivery time of one generated event includes all the time spent on delivering all its multiplied instances.

- Event Filter: receives event sequences and performs match operations on each item of them. A time cost is assigned to the filter and it is a linear function of the sequence length (in our tentative model, filtering is based on

[2] In the case of pull mechanism the channel is requested at the output port for an event that is then requested from the object connected to the input port.

event type matching, other fields of events are neither modelled, therefore nor filtered).

Arbitrary number of instances of the objects can be used in such a simulation, building network configurations of them.

5. Measurements

5.1. Configurations

In our tentative experiments the attributes and parameters were set to model Notification Service implementations based on Java, running on 400 MHz Pentium III computers connected with 10 Mbps Ethernet. Parameters were obtained from [12]. The suppliers timed the event delivery of structured events to the consumers. The need for these measurements arose from the idea that the network management framework used by e.g. a national telecom operator should be based on COS Notification Service, making use of its customizable broadcast capabilities. The configurations tested and the results follow in the subsections below. As mentioned above, the event delivery time of one event included all the time spent in the network by all multiplied instances of that given event. Note that filtering was not applied in the cases of various network configurations (*Figs. 3* and *4*). Modifying the eight-channel configuration with 25 consumers results in network graphs with different supplier-to-consumer-route lengths (i.e. graph depths). Concurrency is examined using these variants. A four-channel configuration was used for various filter configurations.

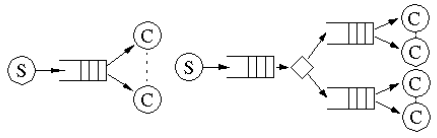


Fig. 3. One and three-channel configurations

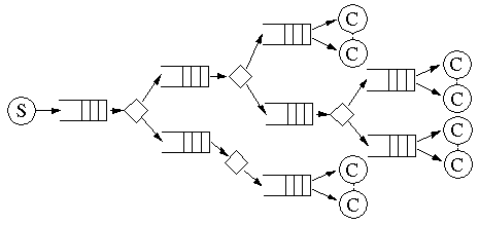


Fig. 4. Eight-channel configuration

One-channel configuration

This configuration included three Suppliers and various number of Consumers using one Notification Channel as the broadcast medium.

Three-channel configuration

This configuration also included three Suppliers and various number of Consumers. The network of the three channels and a Network Node formed Y topology. Events were pushed from the Suppliers to one of the channels, and the node broadcast them to the other two channels. Suppliers were proportionally distributed on the output ports of the last two channels.

Eight-channel configuration

A graph of 8 edges (Notification Channels) and Network Nodes was built. Three Suppliers and various number of Consumers were connected to the nodes selected by chance.

No filtering

Configurations in the cases of various filtering complexity were the same, including three Suppliers and four channels pushing events to various number of Consumers. In the first case, no filtering was applied.

Simple filters

The filter objects were attached to the Notification Channels and the filter constraints set similarly as follows:
$event_type==’A’.
For details on the filter constraint syntax, refer to [8].

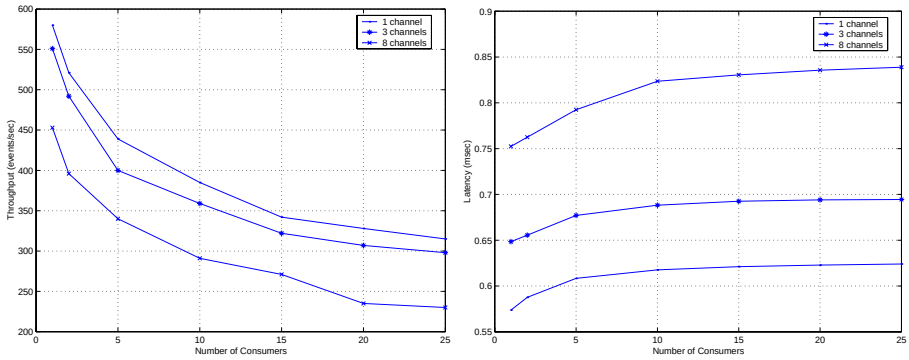
Complex filters

Because filtering is only based on event types in this model, complex filter constraints were set similar to simple constraints:
$event_type==’B’ or $event_type==’C’ and not $event_type==’D’.

5.2. Results and Analysis

Various network configurations

The more consumers are connected to the channel, the lower is the throughput. The reason is evident: the notification channel has to invoke a remote operation on every connected supplier, by which the event is pushed to its destination. At a single channel, these invocations may be executed sequentially or parallel, according to the thread-management of the actual implementation. The more channels are interconnected, the more RPC's are needed for event transfer, so throughput drops. Hence, channels are generally not connected sequentially after another in the notification system, the overall throughput is not inversely proportional to their number. Results for this class of measurements is depicted in *Fig.5*.



(a) Throughput (b) Latency

Fig. 5. Results at various network configurations

Concurrency and scalability issues

Table 1 summarizes how the level of concurrency affects the throughput of the network. As the average graph depth increases, the level of concurrency is degraded, as channels are more likely to be interconnected sequentially. More channels in a route mean more RPCs, so the higher the average graph depth, the lower is the overall throughput of the system.

The throughput figures for various network configurations (*Fig.5a*) provide data for *Table 2*, which indicates the configurations that fulfil the requirement of serving at least 350 events per second. Configurations passing our scalability test

at our settings are marked with √. Note that this requirement in large scale router networks may hit the value of 1000 events/sec.

Table 1. Results at various concurrency levels

Average graph depth	*Throughput (events/sec)*	*Latency (ms)*
1	288	0.792
2	269	0.823
2.5	246	0.830
3.5	235	0.835
4	227	0.838

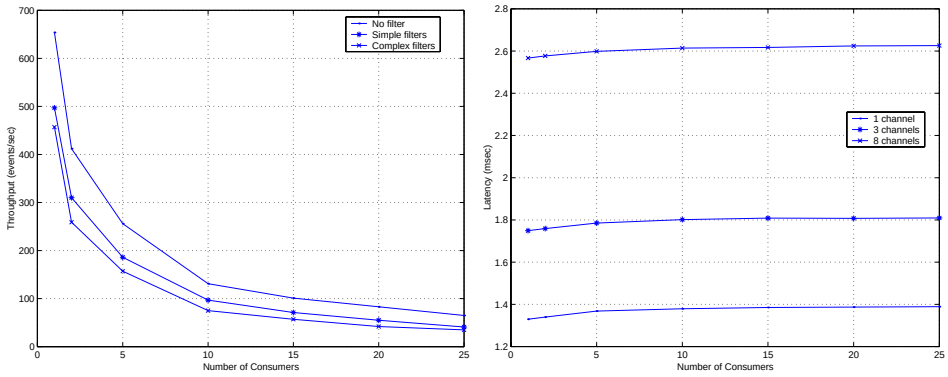
Table 2. Scalability of the channel federation

Number of consumers	*1 channel*	*3 channels*	*8 channels*
1	√	√	√
2	√	√	√
5	√	√	–
10	√	√	–
15	√	√	–
20	–	–	–
25	–	–	–

Various filtering complexity

Event filters use a large amount of computing capacity, which is the bottleneck in notification networks rather than link capacity between network elements or hosts. The reason why throughput decreases as the number of consumers increases has already been discussed in the previous subsection. *Fig. 6* shows that the more complex is the filter constraint, ergo the more CPU-time is needed for filtering, the lower is the overall throughput of the notification system.

The qualitative results of the measurements illustrate the possibility of assessing the performance of an object-oriented distributed system by means of discrete simulation. On the other hand, quantitative results provide useful data for designing the desired configurations.



(a) Throughput (b) Latency

Fig. 6. Results at various filter complexity

6. Simulation of Extended Systems

The tentative model presented above is not suitable for performance evaluation of all CORBA services, it only highlights the possibility and advantages of simulating CORBA systems. Extensions and improvements are being considered and carried out.

6.1. Further Simulated Components

The extended model is capable of simulating traffic on real networks. Background traffic generators are used to generate load on the network. The parameter settings can be customized to model HTTP, FTP, and WAP, etc. protocols, with which IIOP share network resources. The set of objects is also expanded with general clients, servers, ORBs and hosts. Hosts are the computers connected to networks, and they execute arbitrary number of ORBs, according to the actual technology of the modelled CORBA implementation. ORBs behave as transparent communication buses for server and client software components.

Object behaviour is described with scripts written in an IDL-like language, but special objects or standard services, like Naming, Event, Time, etc. may come already encoded. Client scripts include data necessary for specifying the objects to invoke operations on; the distribution-determined speed of request generation, and time costs of function calls. Server scripts include parameters concerning their throughput.

The ORB objects manage network connections, load sharing between servers, security features, error-handling, multi-threading, scheduling and synchronization,

and parameter marshalling and demarshalling. The network connection model is derived from existing TCP/IP network models; links, buffers, switches, and routers are applied. Being simulated objects, the simulated client and server software components collect data in their own measurement attributes, and also share necessary information with the global measurer object that evaluates the overall performance of the network.

In CORBA systems the method invocation mechanism consists of the following steps: (1) Function call on the stub, (2) GIOP message transfer via the communication framework, which may include several message exchanges, and (3) Function call on the object implementation. As mentioned above in Section 4.1, inter-object communication in the simulation software is by means of message transfer. Due to the essential requirement of modelling, being as substantial as possible, all these steps are modelled as message transfer; the message type is the name of the method, and arguments form the message parameters.

6.2. Detail Levels

The model enables three types of simulation.

- ORB-level simulation: Networks, hosts and ORBs are simulated, and their behaviour is described by possibility distributions. This is a GIOP-level simulation.
- Stochastic software component level simulation: ORB-level simulation extended with client and servant software component objects. The behaviour of the objects is also determined by distributions.
- Detailed simulation: Includes all types of objects, their behaviour depends on their actual internal state and stimuli arriving at their input. This enables tests for certain use-cases, exceptional and error situations.

The user of the simulator may choose the type that fits his needs, and complexity and detail level of the desired model.

6.3. Sample Scenario

Architecture

The sample system models the classic CORBA sample application: a bank with a central server managing the customers' accounts, and several branches (clients) around the world initiating transactions (deposits and withdrawals). Branches know only the name of the bank object, they have to retrieve its Inter-Object Reference (IOR) from a Naming Service object. Time Service [5] is also used for the synchronisation scheduling of transactions. The architecture is depicted in *Fig.7*. The *network* transfers messages between *host1, host2* and *host3*. *ORB* objects act as

the middleware interface to the client and server objects: *Naming Service*, *Time Service*, *Bank server* and *Client*.

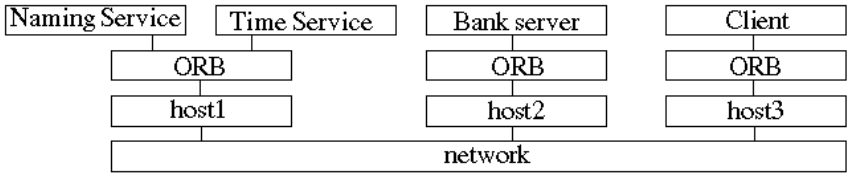


Fig. 7. Sample architecture

This is a detailed simulation (see Section 6.2). The reference application was also developed in order to obtain real parameter values for the simulations. *host1* was a SUN Ultra Enterprise 420R equipped with 4 450MHz UltraSPARC-II processors and 1 GByte of RAM. *host2* and *host3* were SUN Ultra5 computers with a 270MHz UltraSPARC-IIi processor and 128 MByte of RAM, each. The interconnecting *network* was a busy enterprise 100Mbit Ethernet. The operating system was Solaris 5.6. MICO 2.3.1 CORBA implementation [4] with GNU g++ 2.9.5 compiler was applied.

Fig. 8 presents the IDL definition of the used interfaces, and *Fig. 9* an illustrative script that was applied to the model.

```
interface Account
{
   void deposit(in unsigned long amount);
   void withdraw(in unsigned long amount);
   long balance();
};

interface Bank
{
   void new_customer(in string name,
   in unsigned long amount);
   void delete_customer(in string name);
   Account get_customer_account(in string name);
};
```

Fig. 8. Interface definitions

```
bank_ref = nsd->resolv(bank_name);
for(i = 0; i < count1; i++)
{
   mrCustomer = new Customer("MrCustomer");
   bank_ref->new_customer("MrCustomer", amount0);
   nsd->bind("MrCustomer",
   bank_ref->get_customer_account("MrCustomer"));
   for(j = 0; j < count2; j++)
   {
      customer_account = nsd->resolv("MrCustomer");
      customer_account->deposit(amount1);
      customer_account->withdraw(amount2);
   }
   nsd->unbind("MrCustomer");
   bank_ref->delete_customer("MrCustomer");
}
```

Fig. 9. Simulation script of client

Validation and experimental results

Simulation scripts similar to that described in *Fig.9* were executed with the simulator and with a real implementation of the system, as well as invoking several methods of different resource requirements (i.e. resolving an object name in the naming service consumes up to 10 times more CPU time than depositing an account). *Table 3* presents experimental data. Total number of method invocations and execution times, which are assessments on the cases of simulation, are presented.

Table 3. Execution times of scripts

	Execution time(sec)	
Method invocations	*Implementation*	*Simulation*
1000	17.485	17.012
5000	86.312	88.334
10000	172.887	169.960
20000	351.506	355.264

Results validate that the model is suitable for performance assessments of such systems. Maximal difference between real and assessed execution times was *2.71 percent*, which reasonably fulfils the requirements.

7. Conclusions

This paper focuses on a new field of applying discrete event driven simulation: software components of CORBA-based broadcast system are simulated and performance evaluation is carried out.

As a summary we declare that an actual problem was simulated. The model of the Notification Service and its usage in federated channels environment was developed and measurements were carried out. The tentative model represents distributed network management or alarm management system over CORBA. The simulation model is now suitable for performance evaluation of an arbitrary number of notification service implementations and configurations, and can easily be adapted to other services and application objects, based on any CORBA implementations.

The simulation makes it possible to evaluate the performance of large-scale network configurations. There is a certain complexity of the network the distributed system uses, at which performance cannot be evaluated, and behaviour cannot be assessed using pencil and paper queuing analysis. Simulation makes it also possible to deal with various loads on the network using so-called time profiles, e.g. peak hours or transients. The advantage of the method is that it is not necessary to implement and assemble the whole system, which may even grow country-wide. It is easier to take into account the properties of a shared network or the Internet with a sophisticated telecom simulator, where models for non-CORBA communication technologies have been evaluated and used at high reliability in several other applications.

For small scale distributed systems the usual way of performance evaluation is more effective: there is no need for simplifications and/or estimations to build the actual model. Using private or dedicated networks, point-to-point topologies make it also unnecessary to model background traffic on shared resources.

References

[1] BROSE, G., *JacORB Performance Compared*, http://www.inf.fu-berlin.de/~brose/jacorb, Mar. 2000.
[2] Ericsson Telecommunications AB, *PlasmaCORE Programmer's Guide*, Stockholm, Sweden, 1999.
[3] Hewlett-Packard Company, *HP OpenView Communications Notification Server Performance*, http://www.hp.com/ovc/library/np/np.html, 1999.
[4] *MICO - Mico Is COrba*, http://www.icsi.berkeley.edu/~mico, 1998.
[5] Object Management Group, *CORBAservices: Common Object Services Specification*, Revised Edition, Mar. 1995.
[6] Object Management Group, *A Discussion of the Object Management Architecture*, Jan. 1997.
[7] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.2, Feb. 1998.
[8] Object Management Group, *Notification Service*, Joint Revised Submission, Jan. 1998.
[9] SCHMIDT, D. C. – HARRISON, T. H. – O'RYAN, C. – LEVINE, D. L., *The Design and Performance of a Real-time Event Service*, Department of Computer Science, Washington University, St. Louis, MO 63130, USA, 1999.

[10] VEPSTAS, L., *Linux DCE, CORBA and DCOM Guide*, http://linas.org/linux/corba.html, Apr. 2000.
[11] VINOSKI, S., CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, **14**, February 1997.
[12] WILLIAMSON, I., *OpenFusion Notification Service: Performance Evaluation*, A White Paper, PrismTech Corporation, Jan. 2000.