An Application-Oriented Approach for Distributed System Modeling and Simulation

M. Nikolaidou
Department of Informatics
University of Athens
Panepistimiopolis, 15784 70
Athens, Greece
Tel.: (+) 301 – 7275614
Fax: (+) 301 – 7275214
Email: mara@di.uoa.gr

D. Anagnostopoulos
Department of Geography
Harokopion University of Athens
El. Venizelou Str, 17671
Athens, Greece
Tel.: (+) 301 – 9549171
Fax: (+) 301 – 7275214
Email: dimosthe@hua.gr

Abstract

Complexity of applications operating in a network environment has been considerably increased, since numerous architectural models, such as the client/server model and its extensions, have lately emerged. When dealing with distributed applications, network modeling is not so demanding and modeling solutions for widely used network components are already adopted by commercial tools.

In this paper, we introduce a modeling approach for distributed systems, putting the emphasis on distributed applications. This approach enables the analytical description of applications on the basis of predetermined, high-level operations (or actions) which can be customized to conform to specific architectural models. Operations are ultimately expressed in terms of primitive actions. Through this multi-layer decomposition scheme, in-depth analysis of application mechanisms is promoted.

The modeling approach is oriented towards performance evaluation through simulation and a simulation tool has been constructed for this purpose. Modeling examples and a case study for a distributed database banking system are also presented.

1. Introduction

The outburst in network technology gave rise to different types of applications operating in a network environment. Most are based on the client-server model and its extensions, such as the two-tier and three-tier models discussed in [1], and are generally called distributed applications. Distributed applications and the network infrastructure form a distributed system ([2]). Most commercial information systems, such as banking and flight control systems, e-mail and WWW applications, distant learning environments and workflow management systems fall in this category. Development of standards, such as CORBA, allowing the interaction between heterogeneous, autonomous applications, and of programming languages, such as Java, providing native distributed programming support, establish a well-defined platform for distributed application development ([3]).

In current research, a number of cases with different orientation can be referenced. Simulation modeling of customized applications is usually performed in an analytical way, using mathematical models (i.e. the corresponding functions - distributions) to represent network load generation ([4], [5], [6]). Approaches exploiting whether the overall system supports the requirements imposed by specific, customized applications do not emphasize the way applications operate. Application performance exploitation is thus unavoidably depended on the network infrastructure, often overlooking the complexity of application operation mechanisms. Both standardization and complexity increase issues intensify the significance of operation mechanisms. Even though distributed applications depend on the supporting network, application mechanism modeling must be strongly emphasized to carry out an in-depth performance analysis.

In [7], [8], [9] and [10], object-oriented modeling is adopted for network entities and applications. Emphasis is given to networking issues and application modeling is performed at the primitive action layer, using a series of discrete requests for processing, network transfer, etc., in

terms of predefined primitive actions. This, however, cannot be effective, since application decomposition is not supported through a well-defined mechanism. The modeling schemes introduced are not oriented towards specific architectural models and application description is performed rather abstractly. Determination of the effects caused by application operation can not be accomplished without emphasizing the operation mechanisms, making rather improbable to accurate estimate application load. Extendibility and wide applicability, to support variations of the architectural models as well as customized implementations, are also not supported.

Establishing a generic modeling scheme is required due to the heterogeneity encountered in the description of application mechanisms. This scheme must be general to facilitate the representation of different types of applications, i.e. primitive (e.g. FTP) and complex (e.g. distributed databases), according to common modeling principles, and the interaction between applications and the underlying network.

In this paper, we propose a modeling framework for distributed system entities that contributes to the in-depth description of application operation mechanisms. Emphasis is given on the integration of individual guidelines into a generalized modeling framework for distributed systems and not on network modeling, since traditional approaches have already provided effective solutions.

An integrated environment, the Distributed System Simulator (DSS) aiming at the performance evaluation of distributed systems was constructed and is also presented. DSS enables the exploitation of various types of distributed applications, including user-defined ones, as well as the exploitation of the network infrastructure, through its graphical components. Object-oriented modeling is employed for distributed system entities and component preconstruction is supported. Network and application models reside in model libraries. Performance issues are also addressed, since it is critical to guarantee a minimum time delay when simulating an entire distributed system architecture.

Key features of the application modeling scheme introduced are the provision of a multi-layered decomposition mechanism for distributed applications in terms of predefined, primitive actions. Application operation is analyzed on the basis of well-established architectural models, and this process is supported through pre-defining high level operations (action).

The rest of the paper is organized as followed: In section 2, modeling issues are addressed, emphasizing the generalized modeling scheme used to describe distributed applications. DSS architecture is briefly presented in section 3, where extendibility and validation issues are also discussed. A case study, where DSS is used for performance evaluation of a distributed banking system is presented in section 4, while conclusions reside in section 5.

2. Distributed System Modeling

Within the DSS framework, distributed applications are modeled on the basis of the client-server model, consisting of two kinds of interacting processes: clients, which are invoked by users requesting service, and servers, which provide services and are invoked by other processes. Distributed architecture modeling is based on the workstation-server and the processor-pool model, both of which are widely acceptable ([2]). Client processes are executed on workstations, while server processes are executed on dedicated servers or processor pools.

Since distributed systems are multi-entity systems, a modular approach must be employed for the in-depth description of application operation mechanisms. In simulation modeling, modularity often results in a hierarchical structure, according to which components are coupled together to form larger models ([11]). This structure corresponds to the composition scheme depicted in figure 1. Object-oriented modeling provides an almost natural representation of distributed system components.

When extending to elementary (e.g. process) and composite entities (e.g. network node), hierarchical layering enables the construction of complex models through extending the behavior of existing objects and ensures that models of a single entity, organized in a single class hierarchy, are accessed through a common interface, using polymorphism ([12]). Since preconstructed models must correspond to all potential components of a distributed system, composite models, built on elementary ones, must be provided. Implementation of this scheme proves to be notably time-consuming when not supported by automated generation and manipulation capabilities, as the ones provided by DSS. However, it promotes model availability and reduces the time required when composing customized models.

Distributed systems are modeled as a combination of two types of entities: distributed application and network infrastructure entities. Both are described in terms of their elementary components. The network infrastructure consists of nodes, either processing (depicting workstations and processor pools) or relay (active communication devices – e.g. routers), storage devices and communication channels. Distributed applications are described in terms of processes (clients and servers), files and user profiles. Processes and files are elementary components. Files are accessed only through servers of a

specific type (File Servers). User behavior is modeled through User Profiles.

The modeling scheme introduced for the representation of distributed architectures is depicted in figure 1, as a decomposition diagram.

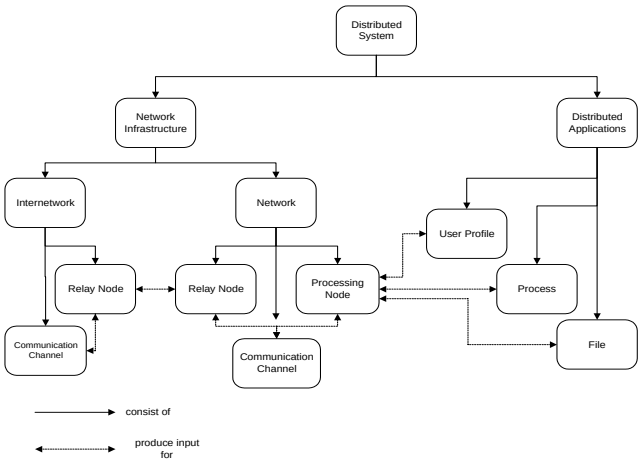

Figure 1. Distributed system decomposition scheme

2.1 Distributed Application Modeling

In most contemporary systems, distributed application operation is based on the client-server model. When designing distributed applications, as indicated in [1], there are many architectural solutions that may be employed regarding the functionality provided by clients and servers and the replication scheme.

In the first generation of distributed applications, functionality was merely embedded in the clients, while servers dealt with data manipulation and consistency issues. This was the heavy client - light server model. Most commercial distributed database applications fall in this category. After the explosion of the Internet and the WWW, this model was no more viable, since functionality was included in Web Servers to minimize communication delay (light client - heavy server model). Furthermore, the aggregate functionality was dispatched into more layers with the use of intermediate ones (middleware) between clients and servers, thus offering common services to clients. In this way, the functionality was enhanced and lighter clients were developed, without interfering with the server. This is the three-tier application model, as opposed to the two-tier model discussed above. Most distributed banking systems fall into this category.

Within the DSS framework, a basic scheme was introduced to facilitate the description of applications, regardless of their complexity and architecture, provided that applications are based on the client-server model. It is thus possible to support any of the above architectures. Two types of processes can be defined: clients, which are invoked by users, and servers, which are invoked by other processes. Access to files is performed through File Servers (FS). The specific interfaces, acting as process activation mechanisms must be defined for each process, along with the operation scenario that corresponds to the invocation of each interface. Each operation scenario comprises the actions that occur upon process activation.

Actions are described through qualitative and quantitative parameters, e.g. the processes being involved and the amount of data sent and received. In most cases, the operation scenario is executed sequentially (each action is performed when the previous one has been completed). However, there are cases where actions must be performed concurrently. This is supported through specifying groups of actions that have the same sequence number.

The predefined actions are the following:

- Processing: indicating data processing
- Request: indicating invocation of a server process
- Write: indicating data storage
- Read: indicating data retrieval
- Transfer: indicating data transfer between processes
- Synchronize: indicating replica synchronization

Each process is executed on a processing node. Processing action indicates invocation of the processing unit of the corresponding node and is characterized by the amount of data to be processed.

According to the client-server model, communication between processes is performed through exchanging messages using request/reply protocols. DSS currently supports RPC, RMI and HTTP protocols. User Interface is responsible for detecting loops in server invocation and resolving them. Request action indicates invocation of a server process and is characterized by the name of the server process, the invoked interface and the amount of data sent and received. It also implies activation of the network, since the request and the reply must be transferred from the invoking to the invoked process, and vice versa. Since group communication is not supported by the existing request/reply protocols, this functionality can be modeled as a parallel execution of multiple requests.

Storing data is performed through File Servers. There are two actions available for data storing, read and write, which are characterized by the amount of data stored and retrieved, respectively, and the file server invoked. Temporary data can also be stored in the local disk, resulting in the invocation of the corresponding node storage element. File Server process supports two interfaces, namely read and write, corresponding to the aforementioned actions.

Transfer action is used to indicate data exchange between processes.

Replication of processes and data is a common practice in distributed applications in order to enhance performance. While process replication is easy to implement, replication of data is accomplished through defining process replicas for handling data and a synchronization policy. Defining such a policy requires solving issues, such as determining the process responsible for the synchronization (the invoking process or a process replica), when synchronization is performed (i.e. each time a change is made or periodically, at pre-specified time points) and the synchronization algorithm.

DSS facilitates the definition of process replicas operating on different nodes and data replicas stored at different file serves. Defining a process or data replica also requires the specification of the synchronization algorithm between process replicas. DSS does not support specific synchronization policies. It allows the description of the logical connection between replicated processes and data during process definition and provides the synchronize action to facilitate the specification of synchronization policy. This action corresponds to the invocation of the synchronize interface, which must be supported by all process replicas. The corresponding operation scenario has to be defined by the user. Synchronize action parameters include the process replicas that must be synchronized and the amount of data transferred.

User behavior is modeled through User Profiles. Each profile includes user requests to the client interfaces that may be invoked by the user. For each profile, execution parameters, such as the execution probability, are also specified. A detailed example of application description using the aforementioned entities is included in section 4.

The actions used to define operation scenarios are either elementary or higher-level ones. In the latter case, they can be decomposed into elementary actions. While processing is an elementary action, write is expressed through simpler ones, i.e. a process and a request sent to File Server. All actions can be ultimately expressed through the three elementary ones, processing, network and diskIO, each indicating invocation of the corresponding infrastructure component. Action decomposition is not performed in a single step. Intermediate stages are introduced to simplify the overall process and to maintain relevant data. The action decomposition scheme is presented in figure 2.

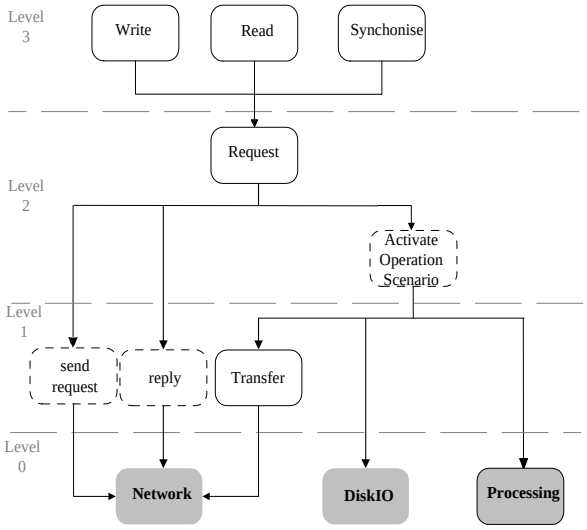

Figure 2. Action decomposition scheme

Dotted rectangles represent intermediate actions, while gray rectangles represent elementary ones. Finally, black rectangles represent the actions used when defining operation scenarios. This diagram can be further extended to include user-defined, domain-oriented actions, which conform to specific architectural models. However alteration or creation of elementary actions is not allowed.

The supported actions are categorized into 4 levels. The lowest level includes only elementary actions, while the highest one includes only actions built upon existing ones. User-defined actions are also placed at this level. Each action can be decomposed into other actions of the same or the lower level. Actions support specific parameters and are derived as ancestors of the action class. During action decomposition, all parameters of the invoked action must be defined. As an example, decomposition rules for request action are presented in figure 3.

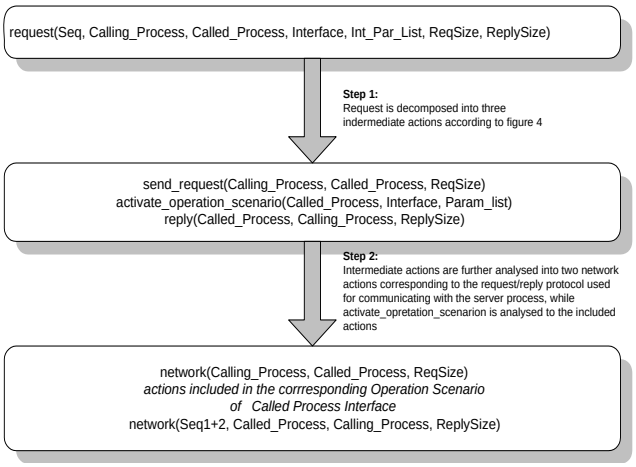

Figure 3. Request action decomposition

2.2. Network Modeling

Modeling solutions for communication network architectures have been employed by commercial simulation environments, as Comnet and OpNet ([10], [13]). For our purposes, we considered the following requirements: uniformity in model description and manipulation, extendibility and improved performance.

In the proposed modeling scheme, network infrastructure is considered as a collection of individual networks and internetworks, exchanging messages through relay nodes (note that networks are here distinguished from internetworks and refer only to local networks). Networks include processing and relay nodes while internetworks include only relay nodes.

Communication element modeling is performed on the basis of a layered scheme, close to the OSI/RM. Although emerging technologies (e.g. ATM) do not fully conform to this model, it serves as a well-established standard providing guidelines for the uniform representation of network entities. The layering scheme enables the description of supported protocols and relations between them through assigning them to one or many layers. In this way, protocol suites (e.g. DARPA TCP/IP, pure ATM) can be easily supported and the interaction between them can be modeled with uniformity.

Key features of the modeling scheme are the following: uniform manipulation of protocol models, capability to either support or not support specific layers, capability to model protocols corresponding to more than one layer or more than one protocol corresponding to a single layer (modeling of protocol suites).

Protocol suites are represented through the communication element entity, which consists of two parts, the peer communication element and the routing communication element. The first corresponds to peer-to-peer protocols (OSI layers 4-7) and the latter to routing protocols (OSI layers 2 and 3). The protocols of the peer communication element have to be common for all processes of the same distributed application.

Processing node entity represents devices acquiring processing capabilities (workstations, servers, etc). Relay node entity represents active network devices, such as routers and switches. Routing devices are modeled as a set of relay nodes linked with each other, each being member of one of the interconnected networks. The addition of a new network to the network infrastructure thus corresponds to the addition of two relay node models.

3. Simulation Environment

Distributed System Simulator was initially developed as part of a distributed architecture design environment, called IDIS ([14]). IDIS is a knowledge-based system that reaches an optimum solution through evaluating a set of alternative architectures. Requirements for network and application modeling, experimentation and model management increased considerably in the late years and DSS evolved into a standalone environment. DSS is based on object-oriented and process-oriented simulation and its current version is implemented using MODSIM III ([15]) for model construction and Java for all other modules.

DSS is modular and includes rule-based modules and a model base, as presented in figure 4. Line connections indicate module invocation and data access.

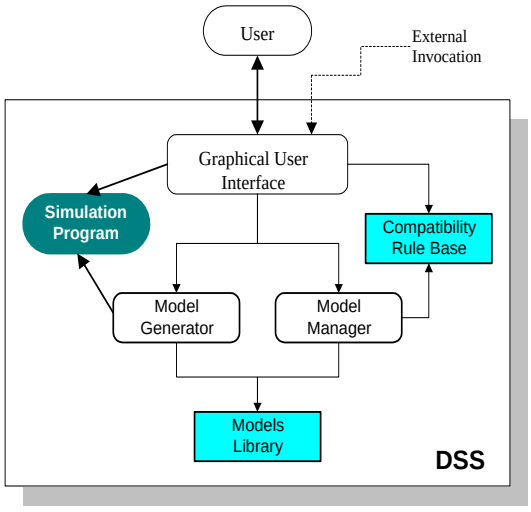

Figure 4. Distributed system simulator architecture

User input involves model and experimentation specifications. Model specifications define the system

under study; experimentation specifications determine how evaluation is performed. DSS constructs the simulation program, using component models that reside in model libraries. Models, either atomic or composite, are implemented as objects organized in object hierarchies. Specification completeness and validity must be pre-ensured, and this is accomplished through rule-based mechanisms.

When experiments have been completed, results are subjected to output analysis to:

1. Determine whether distributed applications operate efficiently. Such measures are average response time and process utilization.
2. Determine the ability of the network infrastructure to support the requirements imposed by distributed applications. Such measures are network throughput, end-to-end delay and internal protocol delay.

3.1. Model Extension and Validation

Extending the supported distributed system component base is a strong requirement for the modeling scheme. User profiles, actions and communication protocols are the most common entity types, new instances of which need to be provided. Model extension is performed through Model Manager and Compatibility Rule Base. In the case of components that provide additional features, such as new actions, these can be constructed on the basis of existing ones (either elementary or not). Construction is accomplished by Model Manager, which establishes a coupling relation between these components. The extension process comprises the following steps:

1. Ensuring model validity and compatibility with the existing models.
2. Inserting component models in the Model Base.
3. Updating Compatibility Rule Base and the Model Manager.

Step 1 is performed using Compatibility Rule Base, while Model Manager performs step 2 and 3.

At the implementation level, the model base is extended using object inheritance. Models are created as ancestors of existing, abstract entity type models. A new action model, for example, would be constructed as a direct descendant of the abstract action model. A concise modeling framework for extending object structures has been in depth described in ([16]).

Model extension and simulation program generation capabilities can only be supported when input specifications are thoroughly examined to ensure model validity. Validation is not trivial, even though models are preconstructed, since models are coupled to form larger ones and are extended to conform to customized implementations. Validation task is carried out through rule-based mechanisms, when specification of networks and internetworks, interfaces and operation scenarios for each process, user profiles, etc., is completed.

The rule base is thus invoked during model customization to ensure that modeling consistency is maintained. Some of the rules included in the rule base are these: at least one interface must be defined for each process, creation or customization of a primitive action is not allowed, etc.

To support the addition of customized models, a graphical environment visualizes the existing model hierarchies. Compatibility Rule Base also ensures that when inserting new models, the existing ones will be in position to reference them, so that the potential coupled models can be formed.

4. Case Study

Distributed System Simulator was used for evaluating the performance of a distributed banking system. Except from headquarters, the bank maintains 64 branches. The banking system supports 24 discrete transactions, grouped in four categories, which are mostly initiated by tellers. The average transaction number of a branch is 500, while the maximum transaction number in central branches is over 1000. The required response time is 15-20 sec for all transactions. Although network infrastructure could be modeled and evaluated using various commercial simulation tools, application description was not possible using the modeling constructs provided and imposed an analysis that gradually extends to the primitive action layer. Except from this, DSS enabled the estimation of the exact amount of data processed and transferred within and between branches.

The banking system has a central database in headquarters, where all transactions are executed, while transaction logs are maintained in local databases at the branches. The central database has 33 stored procedures corresponding to the different execution steps of the 24 transactions. Digital RDB database management system and ACMS are used.

The system architecture is based on the three-tier model and consists of light client applications running on user workstations. The overall network is based on the TCP/IP stack.

Client data are stored locally in the branch file server. When a transaction is executed, the corresponding forms are invoked, each having an average size of 3K. ACMS is invoked up to four times for the execution of the corresponding stored procedure. Before finishing each transaction, a log is stored in the local database.

Server processes that were modeled using DSS are: *File Server* at headquarters and local branches, *CentralDB, LocalDB* and *ACMS*. Since *LocalDB* represents logging, only a simple insert interface had to be implemented for recording the log. *CentralDB* is accessed through the 33 stored procedures, which are implemented and stored in the database. For each stored procedure, a single interface had to be implemented. Since system performance was mainly determined by the interaction of the different system modules and not by the internal database mechanisms, we decided to establish a common representation for all stored procedures, after carefully reviewing the functionality provided by them. A new action called *call_stored_procedure_step* was created and inserted for this purpose in the action hierarchy. Parameters of this action are *preprocessing, data_accessed* and *postprocessing*. *Data_accessed* parameter indicates the amount of data accessed at each step, while *preprocessing* and *postprocessing* parameters indicate the amount of data to be processed before and after access, as a fraction of the accessed data. Using this action, the description of stored procedures was significantly simplified. Each stored procedure consists of one to five steps. The *call_stored_procedure_step* action is implemented as an interface of the CentralDB process in a way similar to read/write and includes the activation of processing, read and write actions. ACMS is modelled as a server process providing the interface *call_ACMS (stored_procedure, inputdata, outputdata, processing*), which initiates the activation of the corresponding stored procedure.

Client applications involve the invocation and processing of forms, the activation of stored procedures through ACMS and log recording. Log recording is depicted through properly invoking the insert interface of *LocalDB*, while stored procedure activation is accomplished through the invocation of the *call_ACMS* interface of *ACMS*. *Form_access (FS, form_name, processing)* was added in the action hierarchy to depict accessing, activating and processing of a form. Using combinations of these three actions, it was possible to describe all applications in a simplified, common way.

Applications were categorized into four groups, each controlled by a different type of user. Applications of same group are not executed simultaneously by the same user. This led us to depict each group as a client process supporting one interface for each specific application. Users are depicted as profiles initiating the corresponding client application.

An example of the application modeling scheme is presented in figure 5.

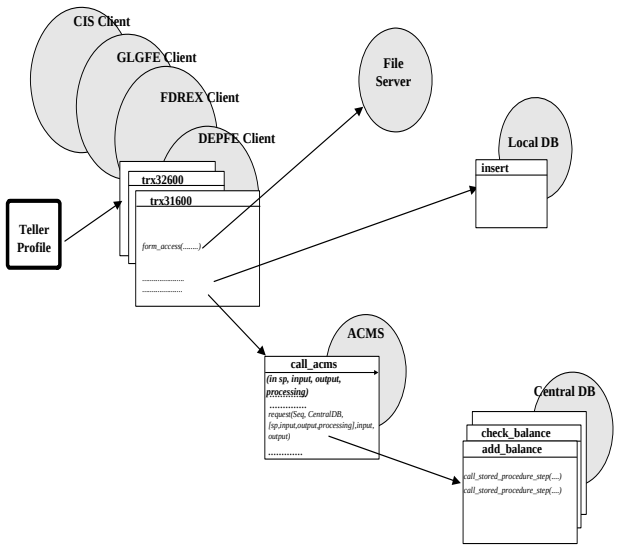

Figure 5. Application modeling example

Modeling advantages offered are summarized as follows: decomposition of application functionality, simplification of the description of client applications, flexibility during process description and detailed modeling.

The capability to extend the action hierarchy was crucial in order to ensure the efficient and detailed application description. If only predefined actions could be used, the same description would have to be repeatedly given for all transactions, e.g. form activation. Furthermore, it facilitated the description of applications at the level of abstraction required by different groups of users.

While the banking system was under deployment, DSS contributed to determining potential weak points and ensuring the response time of client transactions. Since the main activity of all transactions relates to the invocation of the central database through ACMS, special attention was given to the system performance at headquarters. Two drawbacks indicated by DSS were rather critical: First, the processing power of the Central Database Unit was not adequate to execute client transactions within the predefined response time. This proved to be accurate, forcing the bank to upgrade the hardware platform. Second, for the interconnection of branches with headquarters, in-depth load estimation based on the analytical application description suggested that the throughput of specific leased lines should be increased. Ethernet (10BaseT) proved to be efficient for branches since the required throughput was low (less than 0.05 Mbps).

5. Conclusions

Exploring the behavior of distributed systems while emphasizing the description of distributed applications was the objective of DSS construction. Application modeling extends to the operation and interaction mechanisms and conforms to the various forms of the client/server model. Since distributed system architectures are configurable, considerable effort was put into constructing and organizing the appropriate component models to ensure their efficient manipulation.

The modeling approach provides guidelines for modeling the essential, both primitive and composite, distributed system components. The capability to reuse models when implementing customized component models was crucial for the description of different applications, despite the complicated nature of this process.

References

[1] J. Shedletsky, and J. Rofrano, "Application Reference Designs for Distributed Systems", *IBM System Journal*, Vol. 32, No 4, 1993.

[2] Coulouris, G.F., J. Dollimore, and T. Kindberg, *Distributed Systems - Concepts and Design, Third Edition*, Addison Wesley Publishing Company, 2000.

[3] Farley, J., *Java Distributed Computing*, O' Reilly Publishing Company, 1998.

[4] V. Vemuri, "Simulation of a Distributed Processing System: A Case Study", *Simulation Magazine*, May 1991.

[5] R. L. Bagrodia, and C. Shen, "MIDAS: Integrated Design and Simulation of Distributed Systems", *IEEE Transactions on Software Engineering*, Vol. 17, No. 10, October 1991.

[6] V. D. Khoroshevsky, "Modelling of Large-scale Distributed Computer Systems", *Proceedings of IMACS World Congress 1999*, Conf. 15, Vol. 6, 1999.

[7] E. Ginters, Y. Merkuryev, and A. Spungis, "Simulation of Client-Server Distributed Data Processing Systems", *Proceedings of ESM'99*, Budapest, Hungary, June 2-6, 1996.

[8] S., R. Ramesh, "An Object-Oriented Modeling Framework for an Enterprise-Wide Distributed Computer System", *Proceeding of the Americas Conference on Information Systems*, Association for Information Systems, August 1998.

[9] M. Matsushita, M. Ashita, et. al., "Distributed Process Management System based on Object-Centred Process Modeling", *Lecture Notes on Computer Science 0302-9743*, No 1368, Springer Verlag, 1998.

[10] CACI Products, *COMNET III Reference Manual*, San Diego, 1997

[11] B.P. Zeigler, "Object-Oriented Simulation With Hierarchical, Modular Models", copyright by Author, 1995 (originally published by Academic Press, 1990)

[12] D. Anagnostopoulos, M. Nikolaidou, "A Conceptual Methodology for Conducting Faster-than-Real-Time Experiments", *SCS Transactions on Computer Simulation*, Vol. 16, No 2, 1999.

[13] Mil3 Inc, *Opnet Modeler Modeling Manual*, Washington, 1997

[14] M. Nikolaidou, D. Lelis, et. al., "A Discipline Approach towards the Design of Distributed Systems", *IEE Distributed System Engineering Journal*, Vol. 2, No 2, 1995.

[15] CACI Products Company, *MODSIM III The Language of Object-Oriented Programming - Reference Manual*, San Diego, 1999

[16] D. Anagnostopoulos, and M. Nikolaidou, "An Object-Oriented Modelling Methodology for Dynamic Computer Network Simulation", accepted for publication in the *International Journal of Modelling and Simulation.*