Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

A WORKBENCH FOR PREDICTING THE PERFORMANCES OF DISTRIBUTED OBJECT ARCHITECTURES

Sophie Dumas
Georges Gardarin

Laboratoire PRiSM - UniversitÈ de Versailles
45, avenue des Etats-Unis
78035 Versailles Cedex, FRANCE

ABSTRACT

The development of a Distributed Information System (DIS) can lead to critical bottlenecks because of the underlying architecture, which is becoming more and more complex. Todays applications are both object-oriented and based on a new type of three-tiered client/server architecture. In this context, the capabilities of a DIS can be drastically reduced if the performances of the system are not sufficient. Recognizing these trends, industry and research are defining standards and technologies for communicating between components of a DIS and for database access mechanisms. The emerging candidates for these middleware technologies include the OMGs CORBA specification and Microsoft's proprietary solution known as DCOM. A key problem with such complex architectures is the performance issue. This paper presents a simulation-based workbench for predicting the performance of applications relying on these architectures. The proposed tool is based on providing end users with mechanisms to specify the essential characteristics of the application he/she is conceiving and the ability to match the software components with the operational environment (hardware and operating system).

1 INTRODUCTION

When considering a system as a Software Performance Engineering product, one feasible option to be taken into account is the use of simulation techniques especially when analytical models cannot be easily established by simple inspection of the software structure and the operational environment. The main advantage of the performance modeling and simulation approaches is its ability to intervene into the earliest phases of the applicationís life cycle. Moreover, application models can be refined throughout the life cycle phases thus offering a good tool, as far as performance evaluation is concerned, for option validation during the development phase and for supporting activities such as capacity planning and load testing during the operational phase.

As an effort to introduce the principles of the Software Performance Engineering discipline into the universe of object-oriented distributed computing, this paper presents a simulation-based workbench for predicting the performance of applications based on a distributed object architecture, (Bouzeghoub, Gardarin, and Valduriez 1997), (Orfali, Harkey, and Edwards 1996). The proposed tool is based on providing end users with mechanisms to specify the essential characteristics of the application he/she is conceiving and the ability to match the software components with the operational environment (hardware and operating system). The workbench is conceived as a CASE tool for predicting the behaviour of second generation client/server systems that extends the functionality of SMART2, (Ifatec 1996).

The workbench interacts with end users to establish the hardware and software configuration for a distributed object architecture based application to be analysed. By simulating some target application, the workbench provides results related to the communication servers, the usage of CPU, network, disks and the statistics on the execution of user applications. For this last logical component, results are provided at different levels: transaction, program and overall application.

This paper is organised as follows: Section 2 describes the concepts underlying the DCOM architecture and the OLE-DB API. In section 3 the overall workbenchís architecture, modules and functions are presented. The extension to distributed object architectures is explained. Section 4 presents the distributed object architecture modeling paradigm. Finally, section 5 concludes by summarizing the main points of this paper and introducing some future work.

2 OVERVIEW OF A THREE-TIERED ARCHITECTURE BASED ON DCOM AND OLE-DB

A three-tiered architecture distributes object accesses over a *client*, a *processing server* and a *data server*, that means object methods may be processed on a site different from the location site of the object. This section presents DCOM and OLE-DB which are the two bricks to achieve three-tiered architecture using Microsoftís technology. DCOM provides the infrastructure that allow to transparently invoke a remote interface and OLE-DB is the API used to access a DBMS (ORACLE in our study case).

2.1 DCOM Architecture (Distributed Component Object Model)

DCOM (Dcom 1995) is the Microsoft counter part of OMGís CORBA specification (Corba 1995). It is often associated with OLE (Brockschmidt 1993) to achieve Microsoftís distributed document management facility but in this paper we are most interested in Distributed Information Systems and will restrict our focus to solely DCOM.

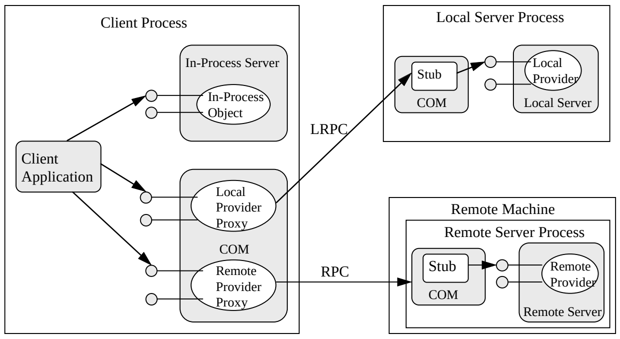

Figure 1: DCOM Architecture

DCOM is designed to allow clients to *tranparently* communicate with objects regardless of where those objects are running (*Figure 1*). Three kinds of servers are distinguished: an *in-process* server loaded into the clientís process space, a *local* server running in a separate process on the same computer as the client, and a *remote* server running on a separate computer. From a clientís point of view, if the object is in-process, the call reaches it directly, with no intervening system-infrastructure code. If the object is out-of-process, then the call first reaches a *proxy* object which generates the appropriate Remote Procedure Call (RPC) to the other process or the other computer. From a serverís point of view, if the object is in-process, the caller is the client itself. Otherwise, the caller is a *stub* object that picks up the RPC from the proxy in the client process and turns it into an interface call to the server object. As far as both clients and servers know, they always communicate directly with some other in-process code.

2.2 OLE-DB (OLE Data Bases) API

OLE-DB is a method for accessing all kind of data via a standard COM interface, regardless of where and how data is stored. It is just an API which role consists in giving applications a uniform access to data stored in DBMS and *non-DBMS* applications (Ole-db 1996). This include storage media such as relational databases, documents, spreadsheets, files, and electronic mail (*Figure 2*). Any component that directly exposes functionality through an OLE-DB interface over a native data format is an OLE-DB *data provider* and a *data consumer* may be a custom program written to one data provider or a generic consumer written to work with a variety of data providers.

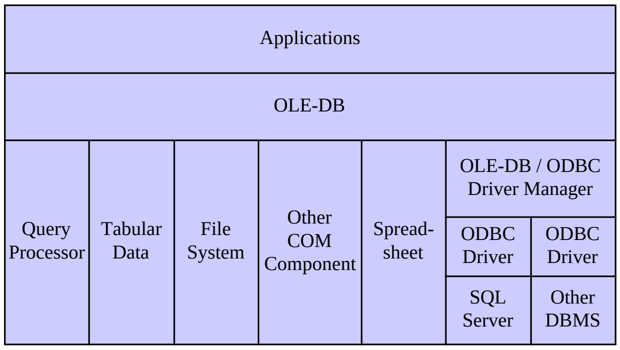

Figure 2: OLE-DB Architecture

The specification introduces seven new object types in supplement to OLE2: DataSource, DBSession, Command, Rowset, Index, ErrorObject, and Transaction. The minimum set of objects and interfaces providers must support is called *base-level interfaces* (*Figure 3*).

Providers must support a *Data Source Object* (*DSO*) which represents a connection to a data source. Then, using IDBCreateSession interface creates a *DBSession* object through which it is possible to access data from a table, create and execute queries, manage transactions, and create a table or an index. And at a minimum, all DBSession objects must support an IOpenRowset interface through which a data consumer generates a *rowset*, making available data from a table.

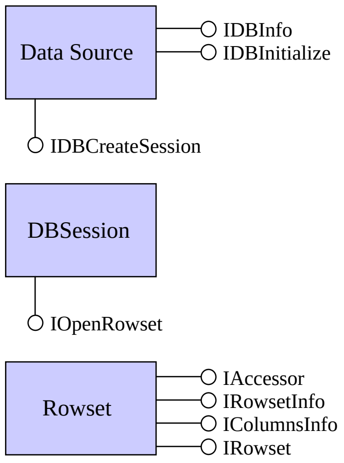

Figure 3: Base Level Interfaces

In our context, from a performance evaluation point of view, we are focusing on the rowset object (*Figure 4*) which is the unifying abstraction that enables all OLE-DB data providers to expose data in tabular form. A basic rowset exposes three interfaces: an accessor (*IAccessor*) providing bindings for application variables, an iterator (*IRowset*) to iterate through the set of rows, and a schema provider (*IColumnsInfo*) returning information about the columns of the rowset. *Handles* are used with accessors to manipulate the contents of the rows. When the row is fetched, the data is *cached* in the OLE-DB component. Note that an analogy can be driven between the limited form of services provided by a rowset and a subset of the services offered by a BOA (Basic Object Adapter) over an object within the CORBA architecture.†

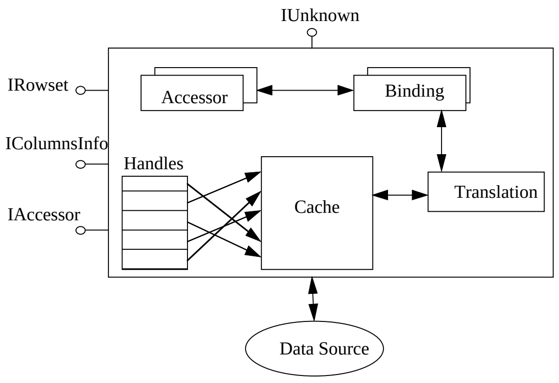

Figure 4: Rowset Object

3 A DISTRIBUTED OBJECT ARCHITECTURE WORKBENCH

The three major approaches for performance evaluation are analytical cost evaluation, simulation, and measurement of prototype systems. Although helpful, each of these techniques is in itself insufficient to predict the performance of a given configuration and select the best arrangement of components. Coupling them in an integrated tool seems to be a promising idea. Thus, our workbench has been first defined and implemented to couple a simulation tool based on queuing networks, an analytical cost model for SQL queries, and a real DBMS optimizer. Then, it has been extended to take into account the distribution of components around CORBA and DCOM.

This section presents first of all the underlying modeling method of the workbench and its functional architecture. Then, it focuses on the extension of this workbench to distributed object architectures and shows all the interactions between the existing models.

3.1 Underlying Modeling Method

The main advantage of the modeling and simulation approach is the possibility to intervene very early within the life cycle of an application, as early as the conception phase. Moreover, application models can be refined throughout the phases of the application life cycle, thus offering a good mean for option validation during the development phase and for supporting activities such as capacity planning and load testing during operational phase. Therefore, depending on the application life cycle phase considered, an end-user of the workbench could be a designer/developper in conception and development phases or a manager in operational phase. Having these premises, the first aim is the one of proposing a method that can be applied in every phase of the software life cycle. To achieve this objective, the proposed method is composed of five steps (Figure 5).

In the *component* definition step, the user can define the essential characteristics of the hardware and software platforms where his/her applications run. Components as processors, controllers and devices, the LAN (Ethernet), IP-routers, versions of DCOM servers, and OLE-DB API are defined. The workbench includes some predefined components for servers (Uniprocessor, SMP -Symmetric Multiprocessing-, Clusters, MPP -Massively Parallel Processing-), disk and drivers (models based on the SCSI or SCSI2 technologies), workstations, and terminals.

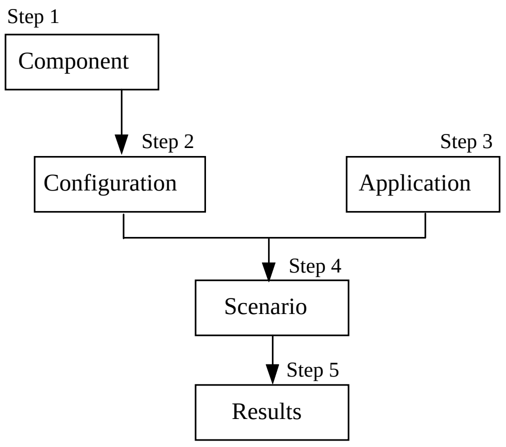

Figure 5: The Prediction Method proposed for the Workbench

Architecture includes hardware and system description and is called *configuration*. The user puts together the hardware and software components to define the final operating environment where his/her *DCOM/OLE-DB* application will be executed.

Application is the step devoted to the software specification and is based on the data description and transaction models. Transactions are specified through a graph-like formalism called *DCOM/OLE-DB Transaction Graphs*. Architecture and application modeling are independent, thus the user can begin by each one at its convenience. Furthermore, this workbench is interesting because the user can evaluate the same applications on different architectures without any change on its application model.

A *scenario* is a merge between a configuration and an application and during this step, the user specifies the entry load of the transactions and locates the data required by the application.

When the user believes his/her scenario is fine, he/she is ready to run a simulation and get some performance *results*. Results are grouped in performance objects (server machines, software servers, transactions, programs, etc.). They give average values or total values for the complete simulation. However, most of them are presented with their confidence interval or standard deviation, to check how relevant is the execution of the simulation. This information can be displayed in different ways and analysed to determine bottlenecks and perform further capacity planning studies for the systems and operational environment.

3.2 Functional Architecture

The workbench is conceived as a Java application interacting with the Oracle 7.3.3 DBMS and with QNAP2V9, an interpreter of QNAP2 (a modeling language with basic type extensions and simulation capabilities) (Simulog 1991).

In the architecture of this workbench (*Figure 6*), the client sends a document containing the application to simulate to the server. In order to verify some constraints and consistency rules, the document is verified on the client side before sending it to the server. When the server receives an application to simulate, it returns to the client a *Unique Simulation Number (USN)* which will be used later to ask for simulation results. Indeed, simulation is *asynchronous* and that is a strong point of this workbench because sometimes a simulation can take a very long time. The communication between the client and the server is done via *Java RMI (Remote Method Invocation)*. When the server receives a document, it generates input files for qnap: **.qnp* files containing the translation of Java objects in QNAP2 macros and **.map* files containing the map of a configuration (how to reach an Oracle instance from a workstation, networks and routers to go through). If the client has SQL queries in its application, the server connects to Oracle to get the *execution plan* of the query and generates a *.qry* file containing the evaluated queries. All these files are given to the simulation engine which generates the results of the simulation using the models defined in the *library*. Then, the results are sent back to the client using the USN.

This workbench is original in the sense that it integrates within a simulation engine real system components, such as the DBMS query optimizer, and analytical cost models used by the cost evaluator. Its major work is to complete the Query Execution Plan (QEP) brought by the DBMS optimizer by adding several statistics concerning its analytical cost. Thus, each entry of a valued QEP (VQEP) is an entry of the original QEP plus the estimated CPU time consumption, the logical I/O requirements, the lock requirements, the estimated data transfer cost between client and server, the volume of data to sort, and the accessed objects (tables, indexes). This VQEP is then passed to the simulation engine. Note that the classical analytical cost models can be extended to object operations (Gardarin 1996).

3.3 Extension to Distributed Object Architectures

Modeling the performances of a distributed object architecture may have several goals. One may want to choose among DCOM and CORBA for example, or to choose among different CORBA implementations (Gokhale, Schmidt, Harrison, and Parulkar 1997). Although there may be early stages of a project where such a choice is still open, more than often performance analysis is targeted for a given middleware and is used to predict performances, to reduce bottlenecks, and to adjust the underlying hardware support. Our goal is to better understand the behavior of a complex Distributed Information System on a given

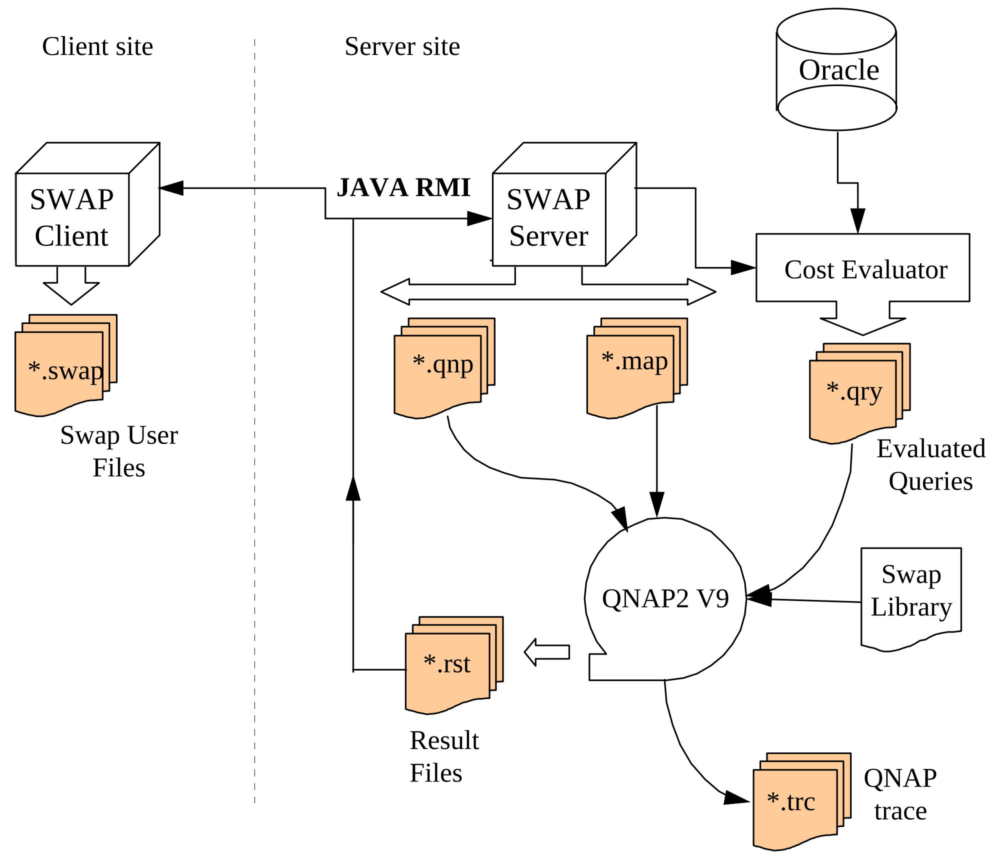

Figure 6: Functional Architecture of the Workbench

platform, with a given architecture, and for a given application. To limit complexity, it is better to highlight the common aspects of CORBA and DCOM: both architectures offer the choice between an in-process server mode, a local server mode, and a remote server mode. Servers are launched through a daemon in CORBA/ORBIX (Orbix 1996) and through a Server Control Manager in DCOM, but after the servers have been started this difference plays no further role. Hence, to take into account the performances of a distributed object architecture (either DCOM or CORBA), the workbench has been extended with a generic model for both CORBA and DCOM†.

A model for OLE-DB has also been developed in order to model the complete behaviour of an application based on a three-tiered architecture. To develop these models, we have used a conceptual simulation modeling method based on queuing networks (Savino, and Puigjaner 1997). This method allows to represent a system as a group of hierarchically interacting simulation models, where each one acts as an agent providing services to the others with no internal knowledge about them.

3.4 Performance Model Hierarchy

The workbench is based on generating a hierarchy of performance models that interact during the simulation process. Some of the models have a static nature, i.e. the behaviour of the components is independent from the user inputs and some others depend totally (either in behaviour as in workload characterisation) on the user inputs.

Figure 7 shows the hierarchy of performance models handled in the workbench. The *hardware* level includes models for the basic hardware component providing processing capabilities. Models for SMP systems, clustering systems, I/O drivers and disks are provided by this layer. The *network* layer includes the models for the basic hardware components providing internetworking capabilities. Models for WANs and Ethernet-based LANs are provided by this layer. The *middleware* layer proposes the needed models for the components aimed at providing the characteristics of a distributed object architecture. The *DCOM* server performance model belong to this layer. A model of the CORBA/ORBIX architecture is also defined at this layer. The *object presentation* layer proposes the model of the OLE-DB API. The *application* layer defines the abstraction of workstation set and terminal set for

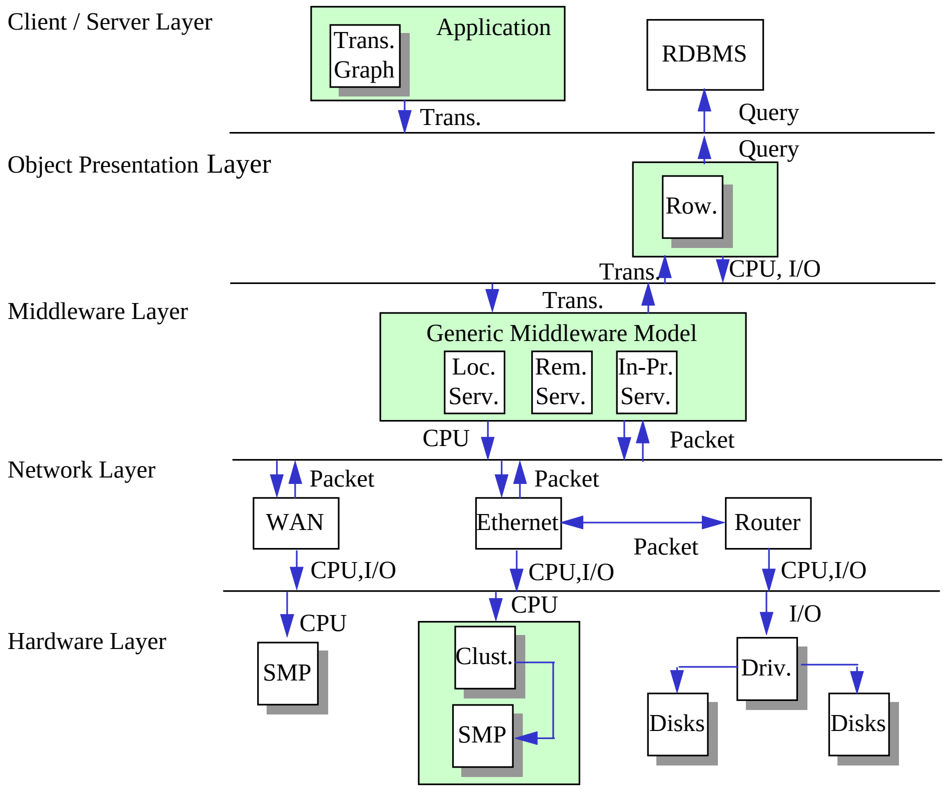

Figure 7: Complete Simulation System

associating a DCOM/OLE-DB application with the workload that they generate. This is a hybrid layer conformed by static performance levels (terminals and workstations) and dynamic performance levels corresponding to the application under study. The main dynamic abstraction introduced by this layer is the *DCOM/OLE-DB Transaction Script*.

4 DISTRIBUTED OBJECT ARCHITECTURES MODELING PARADIGM

This section presents the modeling paradigm of a user application based on distributed object architectures. The impact of this user application structure on the performance models is studied in a hierarchical and bottom-up fashion. The study goes from the lowest level of interaction between application and environment (the basic operations) to the highest level of interaction, i.e. the relationship between the application as a whole and the operational environment on which it runs.

4.1 Abstracting the Operations

At the lowest level, the users are provided with the following operations to model a DCOM/OLE-DB application as follows.

CPU operation: this operation summarises all the operations can be performed in a DCOM/OLE-DB program using the facilities provided by the language used to write it. The affecting performance parameter of this type of operation is the number of instructions to be performed by the CPU on which the application runs.

I/O operation: this operation summarises the I/O operations explicitly issued by the user. The affecting performance parameters of this type of operation are the requested operation (read or write), the size (in bytes) of the stream to be read or written and the identifier of the disk where the operation must be executed.

Thinking Time operation: it represents operations that requests an input from the user and hence a time to wait for. The affecting performance parameter of this type of operation is the time (in seconds) to wait for a user input.

Invocation operation: this operation represents the invocation of a method through a distributed object architecture, either DCOM or CORBA. The affecting performance parameters are the execution context of the server, the called method name with its parameters, their number, their size, and their type.

Rowset operations: these operations can be the **creation** of a rowset, the **insertion** of rows into the rowset, the **deletion** of rows from the rowset, the **retrieval** of rows or the **update** of the rowset. Another operation is the

commit operation to validate the changes produced on the rows of a rowset. For a commit operation, the main affecting performance parameters are the rowset type (sequential or scrollable), the size of its cache and the SQL query. A more precise description of the parameters can be found in (Dumas 1997).

This set of operations could be extended to other operations without any problem. For example, we could add operations concerning the services offered by both DCOM and CORBA (naming, events, persistency).

4.2 Transactions

The next step after conceptualise basic operations is to put together a sequence of operations. In this sense, the first concept a user has to model in a DCOM/OLE-DB application is the *transaction* concept. A transaction can be considered as a list of operations to be performed sequentially during the execution of an application. A DCOM/OLE-DB transaction can therefore be abstractly depicted as:

- T_i, a unique identifier that terms the transaction as an object.
- Seq_i, a sequence of operation instances, that is an operation type with values for each of its parameters. Then, each element on the sequence will be composed by:
  - nb_i (1<= nb <= n), the order of the operation in the sequence (unique for each element of the sequence).
  - $inst_i$, the operation instance itself.

4.3 Transaction Graphs

In order to get an abstract model of a program we have to represent in a suitable way all the execution paths a program can have. In compiling theory, the traditional objects used to represent this situation are the *graphs*. We are not only interested in represent the paths (*static* information) but also its approximate behaviour (*dynamic* information). The mechanism used to represent the program behaviour is based on *probabilities*. From one block B_1 the program execution can continue in the block B_2 with probability *p* if and only if the p*100% of times (the frequency view of a probability measure) the program passes through the block B_1 continues executing the block B_2.

Following the previous reasoning, a DCOM/OLE-DB application can be represented as a graph G whose nodes include DCOM/OLE-DB transactions identifiers. Since the same transaction can appear in different execution paths, the transaction identifier associated to a node is only an attribute and not the node identifier by itself, that is a node in a transaction graph G is a pair (*i*,*t*), where *i* is a unique identifier for the node in G an *t* is a transaction identifier. Arcs in G are weighted with probabilities, i.e. and arc is a pair (*i*,*j*,*p*) where *i* and *j* are node identifiers and *p* is a probability.

Finally, the family $\mathcal{F}$ of graphs that can represent a DCOM/OLE-DB transaction have the following restrictions:

- The empty program must be represented as a transaction graph G with two nodes (0,_) and (∞,_) and one arc (0, ∞, 1). This means that an empty program has an initial point and an ending point and only one execution path from the beginning to the end of the program without execution of DCOM/OLE-DB transactions.
- In any transaction graph G belonging to the $\mathcal{F}$ family, for each node *n* that is the tail of at least one arc in G, the sum of weights of the arcs leaving from *n* must be 1.
- The only node with no exiting arcs is the (∞,_) one.

The following figure shows an example of DCOM/OLE-DB transaction graph. In this figure, the initial and terminal nodes are shadowed. The program may execute only the transaction t_1, only the transaction t_3 or several (one or more) executions of the transaction sequence t_2 - t_4 - t_1 followed by the sequence t_2 - t_4 and many (at least one) t_2 executions; or only the transaction sequence t_2 - t_4 and many (at least one) t_2 executions.

Finally, as a DCOM/OLE-DB application is a set of DCOM/OLE-DB programs, represented as DCOM/OLE-DB transaction graphs, the top level model for DCOM/OLE-DB views a DCOM/OLE-DB application as a set of DCOM/OLE-DB transaction graphs.

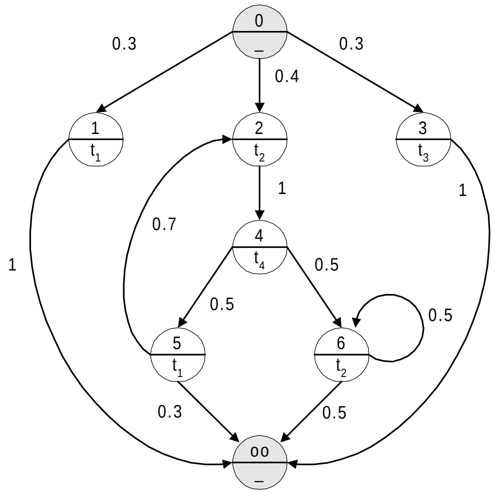

Figure 8: Example of a Valid Transaction Graph

5 CONCLUSION AND FURTHER ACTIVITIES

This paper has presented the architecture and fundamentals of a workbench for predicting the behaviour of applications. The main results obtained with the proposed workbench are:

- A method for predicting the behaviour of applications. The notion of having different phases (component definition, configuration, application and scenario) for defining multiple-level systems provides the user with a separated-of-concerns method for specifying large-scale systems. In addition, the method can be used in every stage of a Software Performance Engineering methodology. It assists designers in making choices on critical issues during the whole lifetime of a project.

- Integration of several approaches such as queuing networks, analytical cost modeling, and real system components, for finely evaluating the performances of an application.

- The extension of the workbench to distributed object architectures, namely DCOM and CORBA. And in order to have a three-tiered architecture environment, we have also integrated the OLE-DB API.

Some models of benchmarking application have been developed to tune the models. Further experiences are ongoing. The ultimate goal of the project is to provide a library of *reusable* components with a runtime environment. The library will ease the development of fine performance models for Distributed Information Systems.

REFERENCES

Bouzeghoub, M., G. Gardarin, and P. Valduriez. 1997. *Object Technology*. Software Engineering series, Thomson Press, Boston.

Brockschmidt, K. 1993. *Inside OLE*. Microsoft Press, Redmond, Washington.

Corba, 1995. *The Common Object Request Broker Architecture: Architecture and Specification*. OMG. Revision 2.0.

Dcom, 1995. *The Component Object Model Specification*. Microsoft Corporation. Draft Version 0.9.

Dumas, S., 1997. *DCOM and OLE-DB model specifications*. ESPRIT Technical Report, HELIOS project (EP 22354), HELIOS/IFA/WP2T2/12.1.97, IFATEC, France.

Gardarin, G., and O. Gardarin. 1996. *Le Client-Serveur*. Eyrolles. Paris, 1996.

Gokhale, A., D. C. Schmidt, T. H. Harrison, and G. Parulkar. 1997. *A High-performance Endsystem Architecture for Real-time CORBA*. IEEE Communications Magazine, Vol. 14, No 2.

Ifatec, 1996. *SMART Userís Guide*. Release 3.0 Beta. Montigny-Le-Bretonneux, France.

Ole-db, 1996. Microsoft Corporation.

Orbix, 1996. IONA Technologies. *The ORBIX Architecture*. Dublin, Irland.

Orfali, R., D. Harkey, and J. Edwards. 1996. *The Essential Distributed Objects Survival guide*. J. Wiley & Sons, New York.

Savino, N., and R. Puigjaner. 1998. An Object-Oriented Approach for the Design of Hierarchical Queuing Network Simulation Models with QNAP2. *Object-Oriented Simulation Conference,* San Diego, California.

Simulog, 1991. *QNAP2 Reference Manual*. Rocquencourt, France.

TPC (Transaction Processing Council), 1996. *TPC Benchmark C Standard Specification*. Revision 3.2.

AUTHOR BIOGRAPHIES

SOPHIE DUMAS is an engineer at IFATEC, a company specialized in technologies around DBMS, Datawarehouse and Internet/Intranet. She is currently working toward a PHD in Computer Science at the University of Versailles. Her research interests include distributed object architectures and performance evaluation of computer systems.

GEORGES GARDARIN got his PHD in 1978 from University of Paris VI. From 1980 to 1990, he was professor at Paris VI University, teaching databases and distributed systems. He was also chief scientist at INRIA where he headed the Sabre project, which was developing an object-relational parallel DBMS. From 1990, he joined the new University of Versailles where he is heading the PRiSM research laboratory, the computer science research laboratory specialized in Parallelism, Networking, DBMSs and Performance Modeling. Georges Gardarin has written more than 80 papers in international journal and conferences, and several books on databases and client-server. He is currently working on federated and semi-structured databases.