Distrib. Syst. Engng 2(1995)102–111

Disciplined approach towards the design of distributed systems

M Nikolaidou†, D Lelis, D Mouzakis and P Georgiadis

Department of Informatics, University of Athens, TYPA Buildings, Panepistimiopolis, 15771 Ilissia, Athens, Greece

Received 27 October 1994, in final form 24 March 1995

Abstract. As the use of distributed systems is spreading widely and relevant applications become more demanding, the efficient design of distributed systems has become a critical issue. To achieve the desirable integration of distributed system components, knowledge from different areas must be combined which leads to increasing complexity. The construction and provision of appropriate software tools may facilitate the design and evaluation of distributed systems architectures. In this paper the architecture and functionality of the Intelligent Distributed System Design tool (IDIS) are presented. IDIS integrates methodologies and techniques from the artificial intelligence and simulation domains, in order to provide a uniform environment for proposing alternative architectural solutions and evaluating their performance.

1. Introduction

The efficient operation of distributed systems (DSs) depends critically on the collaboration of discrete components, such as processing elements, storage devices and communication networks. To achieve maximum utilisation of these components, DSs must be carefully designed. The variety of possible architectural solutions and the integration of knowledge from different areas makes the design a complex task. To effectively explore all aspects of DS construction, software tools have been developed, enabling experts to design DS architectures and evaluate their performance.

Traditional approaches, such as simulation modelling, have been extensively applied during the preceding decade to evaluate the performance of distributed systems and networks. Numerous tools can be referenced, both in the industrial [1, 2] and academic community [3, 4]. Most of them contribute to the behaviour analysis of a predetermined architecture, represented as a simulation model and they cannot make suggestions for the design of the architecture. Software tools have also been built for investigating data or resource allocation problems using formal methods, mathematical models, simulation techniques or, often, a combination of them [5–7]. In the artificial intelligence domain, examples can be found of expert systems for designing single LAN and WAN architectures [8–10]. These systems are based on empirical and experimental rules. Most of them are built to explore certain design issues, such as ELAND [8] that proposes scenarios for the physical topology of local networks, but they cannot be used to evaluate the performance of the proposed solutions.

The research activities presented in this paper are oriented towards the construction of the Intelligent Distributed Systems Design tool (IDIS). IDIS is an expert tool for proposing alternative distributed system architectures according to specifications provided by the operator and available technology. To facilitate the performance evaluation of the proposed solutions, a simulation environment is incorporated into the IDIS framework.

† e-mail: mara@di.uoa.ariadne-t.gr.

2. IDIS objectives

Distributed systems range from a few workstations interconnected by a single LAN to bank or airline systems extended to many remote sites interconnected via WANs. Distributed application requirements include concurrent access to resources by many users, guaranteed response time, service points that are geographically widely distributed, openness and expandability. The main objective of IDIS is to assist DS developers in the decision making process during the design of a new system or the reconfiguration of an existing system, taking into account distributed application requirements. IDIS does not provide commercial solutions. It proposes alternative scenarios for the configuration of the system and data and process placement. Since the performance of a DS depends critically on the performance of the network infrastructure, special attention is given to the protocols used to support the distributed applications and the network topology design.

The distributed applications described by IDIS are considered to reside in different *locations*. A location is defined as a region in which the range does not exceed 2 km, ensuring that all resources within a location can be physically connected through a LAN (the range of sites interconnected by a fibre backbone is limited to 2 km).

0967-1846/95/020102+10$19.50

Given that the scale of DSs varies from single LANs to worldwide applications, IDIS should enable the operator to define the range of the system according to his/her needs; thus, both the definition of locations as well as the specification of their size are performed under the operator's control. If, for example, an IDIS operator describes the sales database of a small firm, the locations will most likely be different floors corresponding to the departments in the firm. If he/she is describing the information system of an industrial complex, the locations can be floors in the main corporate building, buildings in the corporate park and remote branch offices in different cities.

Locations are used to define the access points of a DS. Distributed applications consist of interacting processes. An IDIS operator also specifies the processes operating in it along with a formal description of their operation. This is expressed in terms of processing, exchanging and storing information. There are two kind of processes, front-end processes which are invoked by users and back-end processes, which are invoked by other processes. Data storage is possibly only through system back-end processes called file servers (FS). The operator also specifies the profiles of all the users in the distributed environment. Profile description includes the front-end processes invoked by the user, the probability of their invocation and the location of the user. Process placement is accomplished by IDIS or, alternatively, by the IDIS operator.

To realise its goals, IDIS aims at the:

- placement of processes operating in the distributed environment in order to minimise network traffic
- design of the network infrastructure in order to satisfy the requirements imposed by the distributed applications
- evaluation of the system performance.

The configuration of the DS is formed based on the integration of the workstation-server and processor pull model. A workstation is allocated to each user for the execution of front-end processes invoked by him/her. Back-end processes are executed on dedicated processing nodes. Their architecture is defined by IDIS. IDIS also proposes the network architecture for the interconnection of nodes within locations and the location interconnection schema. IDIS is based on the OSI/ISO RM in order to specify the network architecture. The RPC mechanism is used to describe the three upper layers. RPC is the most common protocol for interprocess communication in DSs. For the description of the four lower layer of OSI RM, IDIS supports a variety of protocols for the implementation of LANs and WANs. All protocols are selected because they are commercial and organisation standards, and are considered to fulfil current and future communication needs. Network interconnection is carried out according to OSI in the data link and network layer via bridges and gateways. The protocols supported by IDIS for all the OSI layers are presented in Table 1.

The tasks that must be accomplished by IDIS cannot be achieved using algorithmic methodologies. In spite of this, methods exist which offer partially satisfactory answers to the problems IDIS is dealing with. The time required for the exhaustive combinatorial examination of all alternatives can be drastically reduced by the introduction of empirical criteria (rules of thumb usually used by DS designers; the data and process allocation problem is considered NP-complete [7]). A DS configuration problem that involves several complex interrelated issues can be solved using heuristic techniques. Furthermore, since most of these problems are semi-structural, the expert system approach was considered as the most appropriate for IDIS implementation [11]. Thus, IDIS serves as a tool of formal description and exploitation of various distributed system design methods and techniques based on empirical knowledge and algorithmic methodologies.

Table 1. Protocols supported by IDIS and their relation with the OSI RM.

OSI Protocol Stack	Protocols Supported By IDIS					
Application						
Presentation	RPCs					
Session						
Transport	DARPA TCP	DARPA UDP		HSTP	ISO TP4	VMTP
Network		X.25	IP	DARPA IP		
Data Link	B-	HDLC		ISO LLC (IEEE 82.2)		
	ISDN	(X.25 Layer 2)		IEEE	IEEE	ANSI
Physical	ATM	X.21	V.35	802.3	802.5	FDDI

The output of IDIS is a detailed parametric description of the functional and performance characteristics for all resources included in each location, processes operating on it, and protocol-stacks used to describe networks and internetworks. The overall performance of the proposed architecture can be explored under actual conditions using simulation tools. Examination of the DS behaviour under conditions imposed by the described applications may determine the existence of bottlenecks and low resource utilisation. The process view simulation approach is adopted and automated program generation capabilities are incorporated into IDIS.

3. IDIS architecture

IDIS consists of two individual subtools, an expert system called the Architecture Definition Tool (ADT) and a simulation environment called the Performance Evaluation Tool (PET), which operate independently and collaborate as shown in Figure 1. The ADT maintains the necessary knowledge to define the architecture of the distributed system and the PET evaluates the proposed architecture performance.

The ADT consists of three modules, co-ordinated by a fourth one, called the *Manager*, which is responsible for

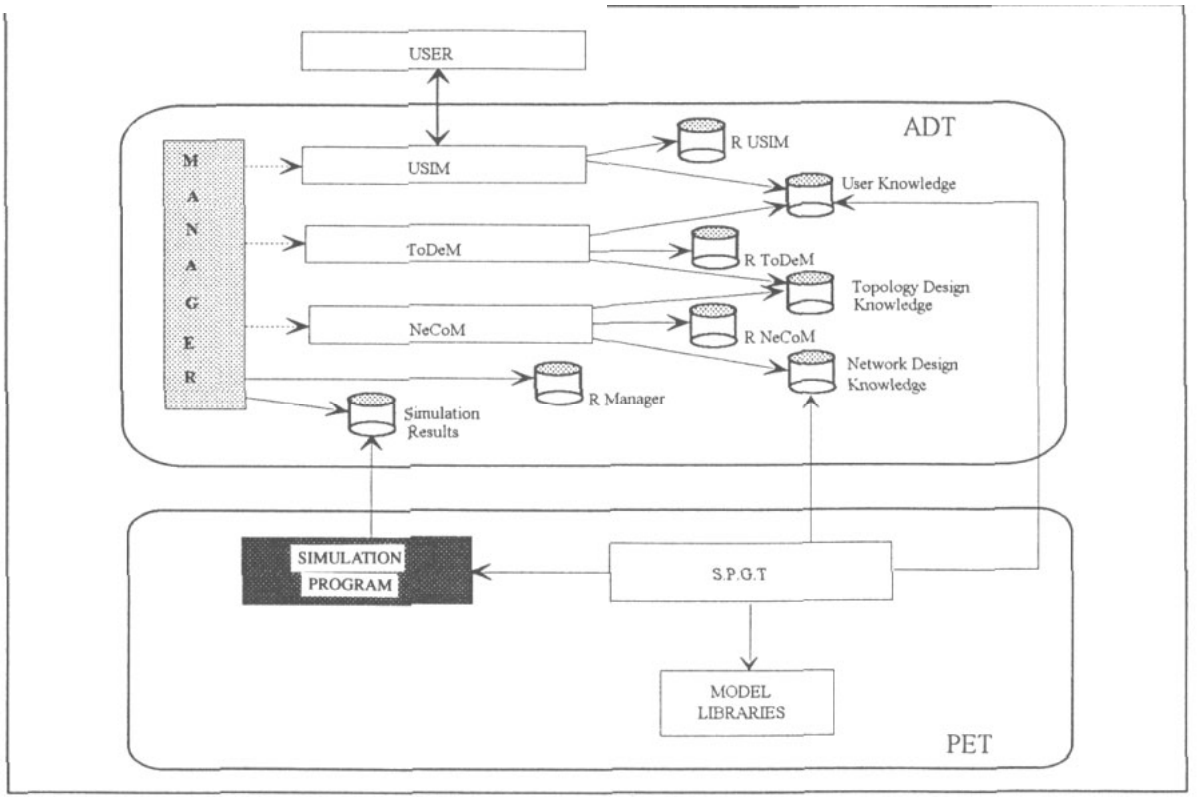

Figure 1. IDIS architecture.

the invocation of all others. The modules are invoked in the following order:

- *User Interface Module (USIM):* The User Interface Module enables an IDIS operator to define the locations on which distributed applications are executed, and to describe their operation. He/she also provides control information concerning specific conditions for the performance evaluation of the system. The user interface is fully system guided to avoid any inconsistency, contradiction and incoherence, according to rules residing in the *RUSIM*. The part of the Knowledge Base containing the information obtained by an IDIS operator is called *User Knowledge*.
- *Topology Design Module (ToDeM):* The Topology Design Module is responsible for process placement (user defined and File Servers) and the construction of the internetwork topology. ToDeM also determines the internetwork type (LAN or WAN) and instantiates the values of the parameters describing the network infrastructure characteristics. These parameters indicate the maximum application requirements imposed on network resources. As indicated in Figure 1, the ToDeM explores two parts of the Knowledge Base, the User Knowledge and a set of rules, called *RToDeM*, which consist of formal descriptions of experimental, mathematical and empirical techniques for data and process placement and topology design. This knowledge is permanently stored in the Knowledge Base and can be subjected to updates through IDIS proper mechanisms. The exploration of knowledge by the ToDeM produces the fact base *Topology Design Knowledge*, which is incorporated into the Knowledge Base.
- *Network Construction Module (NeCoM):* The Network Construction Module designs the network infrastructure using User Knowledge, Topology Design Knowledge and a set of rules, named *RNeCoM*. RNeCoM contains rules for protocol selection and combination and for the assignment of network resources to processes. It is part of the Knowledge Base and can be updated by IDIS proper mechanisms. The NeCoM builds the network infrastructure in order to satisfy the maximum user requirements. The proposed network architecture must conform with the user specifications without wasting valuable processing or communication resources. If the NeCoM cannot satisfy user requirements, it suggests the best acceptable solution. The exploration of knowledge performed by the NeCoM results in the fact base *Network Design Knowledge*, incorporated into the Knowledge Base. This knowledge is used by PET for the construction of the simulation model in order to evaluate the proposed architecture performance.
- *Manager:* The Manager is responsible for co-ordinating the ADT operation. All the modules are invoked by the Manager according to metarules residing in *RManager*, which orientate the ADT inference engine during the exploration of the Knowledge Base. The Manager is also responsible for determining the completion of an IDIS operation, when all the user requirements are satisfied or the ADT Knowledge Base is exhausted. To determine if the user requirements are satisfied, the Manager examines the simulation results, which are stored in the fact base *Simulation Results* after the completion of the simulation process.

PET is invoked when the ADT has completed the design of the network infrastructure and is responsible

for estimating its performance according to the user specifications. It consists of the following two modules:

- *SPGT:* The Simulation Program Generation Tool transforms model and program specifications to simulation programs through the use of the Model Libraries. The SPGT imports DS component models and constructs the simulation program according to the knowledge produced by the ADT to describe the DS model and the control data concerning the experimentation process.
- *Model Libraries:* Model Libraries are constructed to facilitate storing and retrieving DS component models. Models are combined during the simulation program construction phase and form a larger composite model. They can be viewed as object classes, organised in hierarchies due to the advantages offered. Models are directly imported by the SPGT, when building the overall DS model.

4. Architecture Definition Tool

As previously mentioned, the Architecture Definition Tool (ADT) consists of three independent modules exchanging information through a common Knowledge Base. To implement IDIS objectives, the ADT must be able to store and evaluate empirical information. It also must facilitate the conditional invocation of modules, the interruption of a module operation and the invocation of another, and the maintenance of intermediate results. For all these reasons, the blackboard architecture [12] was chosen for ADT implementation. The introduction of the Manager allows the invocation of independent inference engines consulting specific parts of the Knowledge Base for the completion of an autonomous operation. The same architecture is also used for the implementation of all the modules. For the implementation, the Prolog programming language is used, since it offers a uniform environment for the development of all modules.

4.1. Manager

The Manager is responsible for the invocation and co-ordination of all modules as well as the evaluation of the simulation results. The operation of the ADT is supervised by the Manager and includes the following steps:

(1) The USIM is invoked to obtain the description of the distributed applications and their requirements from the IDIS operator.
(2) The ToDeM is invoked to place back-end processes and File Servers and to construct the network topology.
(3) The NeCoM is invoked to instantiate the network resources.
(4) The SPGT is invoked to generate the simulation code.
(5) After the simulation process is completed, simulation results are evaluated. If all the operator's requirements are satisfied, the USIM is invoked to present the results to the IDIS operator and intermediate knowledge is removed. If application requirements are not satisfied or the utilisation of the network resources exceeds the proper limits [13], the NeCoM and ToDeM are selectively reactivated under the supervision of empirical redesign rules residing in *RManager* and step 4 is reactivated. If there are no alternative solutions and the IDIS operator requirements are still not satisfied, the most efficient solution is presented.

The completion of the simulation phase is the most time consuming part of the IDIS operation. To accelerate IDIS performance, redesign metarules are invoked by the Manager so that all possible changes are carried out before the simulation process is reactivated.

4.2. User interface

The User Interface Module (USIM) is a fully system driven environment. The IDIS operator explicitly provides the information requested in a predefined form. The operator is responsible for the complete description of the system specifications, and the USIM is responsible for testing the correctness of the description. User information is stored in the Knowledge Base as presented in Figure 2.

During the description of the applications, the USIM is responsible for ensuring that the information provided by the IDIS operator conforms with the predefined structure and form, as well as checking for any possible contradictions and omissions (knowledge acquisition control). For example there must be a *processing( )* predicate for a back-end process each time it is invoked by a *request( )* predicate. USIM also extracts possible implicit knowledge from IDIS operator descriptions. For example, for each *diskIO( )* predicate, a *request( )* predicate is constructed, indicating information exchange between the process and a File Server as well as a *processing( )* predicate for the File Server.

The User Interface Module is also responsible for the presentation of the proposed architecture and the expected performance characteristics to the IDIS operator.

4.3. Topology design

As previously mentioned, the primary goal of the Topology Design Module (ToDeM) is to suggest locations for processes and data and to define the way in which networks corresponding to locations should be interconnected. The secondary goal is to estimate the application requirements from the network resources. As a result, the ToDeM operation is divided in two parts. The first part corresponds to process placement and the design of the internetwork topology. The second concerns the instantiation of the parameter values, describing the network infrastructure characteristics.

Topology design is based on the following assumptions:

(1) Resources in the same location are connected via LAN protocols.

```
location (LocCode)                                  /* locations are specified by a unique name */
locdist (LocA, LocB, Distance)                      /* distance between locations */
application (ApplCode, BackEndProcessList,          /* application are specified by a unique name */
        FrontEndProcessList)                        /* for each application back-end and front-end */
                                                    /* processes are specified uniquely */
frontend_parameters (FrontEndCode, ResponseTime)    /* front-end process parameters */
user_parameters(ProfileID, Location, StartTime, EndTime,   /* user profile parameters */
        Variance, MeanRequestIntTime, Users,
        FrontEndProcessList)
shared_data(ID, ProfileList)                        /* user profiles using shareable data */
request(ProfileID, SourceLoc, SourceProcess,        /* information amount exchanged */
        DestLoc, DestProcess, RequestAmount,        /* between processes */
        ResponseAmount, Kind, Shareable)
processing(ProfileID, Location, Process, Amount)    /* information amount needed to be processed */
                                                    /* by each process */
terminal(ProfileID, Location, Process, Amount)
diskIO(ProfileID, Location, Process,                /* information amount needed to be stored */
        Amount, Kind, Shareable)                    /* by each process */
```

Figure 2. Predicates representing User Knowledge.

(2) Locations are connected via LANs, when the distance between them is less that 2 km.
(3) Front-end process replicants are placed in all the locations, where corresponding user profiles are defined.
(4) Back-end processes can be replicated. Processes are obliged to use the same back-end replicant, if they share common data.
(5) File Servers are placed on all the locations with back-end process replicants and interactive applications.
(6) At least one File Server must be placed in each LAN internetwork.

The algorithm for the back-end process placement is based on the avoidance of unnecessary data transfer between locations. The algorithm is presented in the following manner:

Step 1: For each back-end process find the profiles calling it, that are using shareable data.
Step 2: For each profile set construct a different back-end process replicant to place the replicant:
  (i) Find the sets of locations that can be interconnected via an LAN. (If a set contains only one location, the location can not be interconnected via an LAN).
  (ii) Find the average network throughput caused in each location from data transfer involving profiles using shareable data.
  (iii) Find the average network throughput caused in all possible LAN internetworks.
  (iv) Place the back-end process replicant in the LAN internetwork that has the maximum network load.
Step 3: For each LAN internetwork with one or more back-end process replicants, place one back-end process replicant in the location having the maximum load.

File Server placement is accomplished in order to satisfy the application response time and to distribute equally the internetwork average load. The algorithm for File Server placement is similar to that used for back-end process placement.

The second phase of ToDeM operation instantiates the values of parameters considered as necessary to determine the DS configuration. This is an intermediate phase, which transforms the information given by the IDIS operator to that needed by the NeCoM to construct the network infrastructure. The output of the ToDeM second phase is incorporated in the Topology Design Knowledge and consists of the predicates presented in Figure 3.

Predicates *networkReq( )* contains parameters determining the requirements for networks and internetworks.

```
networkReq(ID,LocList,MaxDistance,MaxAvgMessage,MaxUsers,JobsCovert,MaxThroughput,KindOfThroughput)
commonNetReq(ID, CommonNetList, JobsCovert, MaxAvgMessage)
relayNodeReq(ID,Loc, InterNetworkList, MaxThroughput)
processReq(ID, Loc, Proc, MaxProcThroughput, MaxNetThroughput)
commonNodeReq(ID, Loc, ProcessesList, MaxUsers)
diskReq(ID,Loc, MaxThroughput, Amount)
```

Figure 3. Predicates representing the application requirements from the network infrastructure.

Applications executed at different locations need a uniform network environment in order to communicate. Since in its current state the ADT builds protocol stacks containing one protocol per layer, common protocols must be used in all locations for end-to-end communication between process. These protocols correspond to upper OSI layers (4–7) and must satisfy common parameters contained in predicates *commonNetReq( )*. Predicates *relayNodeReq( )* indicate the existence of routing elements interconnecting networks. Each location network communicates with others via one or more internetworks. To establish interconnection, one relay node is defined per location.

Predicates *processReq( )* contain parameters indicating the user requirements for the processes running in each location, representing the maximum throughput needed for data processing (byte/s) and maximum throughput needed for data transfer processing (byte/s). Predicates *commonNodeReq( )* indicate processes that can be executed in the same processing node (i.e. they are not simultaneous). Predicates *diskReq( )* contain the requirements imposed on data storage in each location. The access to data storage devices is enabled through File Servers.

The ToDeM can be conditionally invoked by the Manager in order to place back-end process replicants or File Servers in specific locations defined by redesign metarules residing in *RManager*. The ToDeM is responsible for determining any changes in the location interconnection schema and for estimating the values of parameters representing the application requirements from the network infrastructure. Thus the first phase of ToDeM operation can be executed conditionally, while the second phase cannot.

4.4. Network design

The Network Construction Module (NeCoM) formulates the proposed DS architecture based on the workstation-server model, taking into account its extensions as the processor pull model. NeCoM operation consists of the following phases:

(1) Communication Element design for all networks and internetworks
(2) Relay Processing Node design for each location
(3) Processing Node design for each location
(4) Storage Device design for each location.

During the first phase, Communication Element Design, the NeCoM constructs the protocol stacks corresponding to networks and internetworks. Communication Elements are divided in two parts, the Peer Communication Element which corresponds to peer-to-peer protocols (OSI layers 4–7) and the Routing Communication Element which corresponds to point-to-point protocols (OSI layers 1–3). The NeCoM forms the protocol stacks by choosing the protocol corresponding to each layer and determining protocol parameters, such as data unit size, window size, processing delay, etc. Protocol selection is performed top-down, specifying first the transport protocol parameters, then the network protocol parameters, and so on. The whole process is performed under the supervision of metarules ensuring protocol compatibility (i.e. if X.25 is selected in the Network layer, HDLC must be selected in the Data Link layer also), and taking into account limitation and/or specifications imposed by upper layer protocols. NeCoM first constructs the common Peer Communication Elements used for the description of networks which are involved in the execution of applications demanding a uniform network infrastructure. These networks are specified by *commonNetReq( )* predicates, as indicated in Section 4.3. Communication Elements corresponding to location LANs are then structured using the common Peer Communication Elements whenever this is indicated. The Routing Communication Elements are constructed using DARPA IP and local area protocols. Finally, the Communication Elements corresponding to internetworks are formed. They only include Routing Communication Elements corresponding to protocol combinations for LANs and WANs.

In the second phase, Relay Node characteristics included in networks and internetworks are determined according to the *relayNodeReq( )* predicates and the configuration of the connected networks. If two IP networks, for example, are interconnected via a public X.25 internetwork, the relay nodes included in each network must enable the encapsulation of IP packets so that they can travel through an X.25 connection.

Processing Node and Storage Device characteristics are determined in phases 3 and 4. The NeCoM does not decide the type of workstation to be used, but it suggests the processing power needed and determines the processes which can be served by a specific processing node. The NeCoM assumes that all front-end processes can be executed in the user's workstation, but back-end processes are executed at specialised server workstations. Nodes, processing as well as relay, support the protocol stacks described by the Communication Elements of the network or internetwork they belong to.

The NeCoM results are stored in the fact base Network Design Knowledge, incorporated in the Knowledge Base; it is presented in Figure 4.

The NeCoM can be conditionally invoked by the Manager in order to increase the network resource

```
communicationElement(Instance, [LocationList], [PeerComID, RoutComID]).
relayProcessingNode(Instance, [Location, InternerworkList, ProcessingPower], []).
processingNode(Instance, [Location, ProcessList, ProcessingPower, Number], []).
storageDevice(Instance, [Location, Capacity, Speed], []).
```

Figure 4. Predicates representing Network Design Knowledge.

performance or to change the configuration (supported protocol stack) of specific networks or internetworks. The NeCoM is built according to the blackboard architecture, thus the inference engines corresponding to its operation phases can be independently invoked by the redesign rules of *RManager*.

5. Performance Evaluation Tool

As previously mentioned, the Performance Evaluation Tool (PET) is responsible for evaluating the architecture proposed by the NeCoM. PET operation includes two modules:

5.1. Simulation Program Generation Tool

Automated program generation proved to be an efficient technique for dealing with the numerous combinations encountered when designing the proposed architecture that ought to be represented as a simulation model. As illustrated in Figure 1, a module called the Simulation Program Generation Tool (SPGT) is incorporated within the IDIS framework to facilitate the construction of complex DS models. The SPGT performs the automated specification transformation into program code. Specifications reside in the Knowledge Base and refer either to the model construction (i.e. combining individual DS component submodels in order to form a single composite model) or to the actual simulation experiment (control data referring to the execution of the simulation experiment). The SPGT extracts specifications through accessing different parts of the Knowledge Base and builds the simulation program based on the pre-constructed submodels residing in the Model Libraries. Models are imported only through their name; the SPGT is not involved with their implementation, which contributes to the latitude and flexibility of the whole process.

Although most generators in the simulation area are considered to be interactive, the SPGT has no interaction with the IDIS operator. Instead, all specifications provided by the IDIS operator demand a considerable degree of processing by the ADT modules before they are transformed to simulation specifications.

The SPGT is implemented in Prolog. The language selected for model implementation purposes is MODSIM [14], which is a high-level, object-oriented language for performing simulation experiments. DS components models are therefore implemented as MODSIM objects.

5.2. Model libraries

Use of the ADT may result in numerous combinations of individual DS components in forming the overall system architecture. The PET should be in position to deal with all different scenarios; thus, it should acquire characteristics such as modularity and extendibility to deal with the complexity encountered. Process oriented simulation was considered to be the most appropriate methodology for conducting simulation studies in this domain, since DS components and data exchange between them may be obviously represented as a model consisting of interacting objects. These objects reside in the Model Libraries, organised in class hierarchies in order to achieve inheritance capabilities.

For the purpose of the simulation study, the DS architecture proposed by the ADT is a detailed description of the network infrastructure, supporting the operation of the distributed application, and distributed applications specifications, which are originally provided by the IDIS operator and are not altered while being processed by the ADT.

The network infrastructure is viewed as a collection of individual networks and internetworks interacting through message interchange. Each network (or internetwork) consists of network nodes (processing and relay). Processing nodes are represented as individual objects consisting of two major parts: the processing element and the communication element. Based on the NeCoM description, the communication element of the node entity is viewed as a set of discrete layers and each layer supports a single protocol. Owing to their common characteristics, protocols corresponding to the same layer are organised in class hierarchies. The processing element of the node is viewed as a processing unit, considered to act as a CPU, but no modelling of the processing activity is actually required. Every processing node may therefore be modelled as an object having as components individual protocols and a processing unit.

Relay Nodes are represented as individual objects consisting of two major parts: the relay element and the communication element. To achieve interconnection, specific protocols for the relevant OSI layer have to be supported in order to achieve message interchange. Relay element modelling explicitly requires the provision of these protocol models, which occasionally turns out to be a drawback since interconnection mechanisms are not yet standardised.

The communication channel, as a basic component of every network of internetwork, is also modelled as an object. The most important channel characteristics for simulation purposes, namely throughput and error rate, are represented as object properties.

Distributed applications are modelled as individual objects, organised in class hierarchies. Each application model represents a set of processes (front-end and back-end) as described by the IDIS operator. Each process corresponds to a processing node, that is, there should be a single process feeding with input data a single processing node, since there are no simultaneous processes operating on the same node according to the NeCoM description. Input data consist of requests for processing, exchanging and storing information used by the ADT to describe the functionality of distributed applications. The method by which input data are generated and forwarded to the processing nodes is characterised by the statistical distributions defined by the IDIS operator during user profile description.

The above observations may be summarised as follows: the overall DS model consists of a network

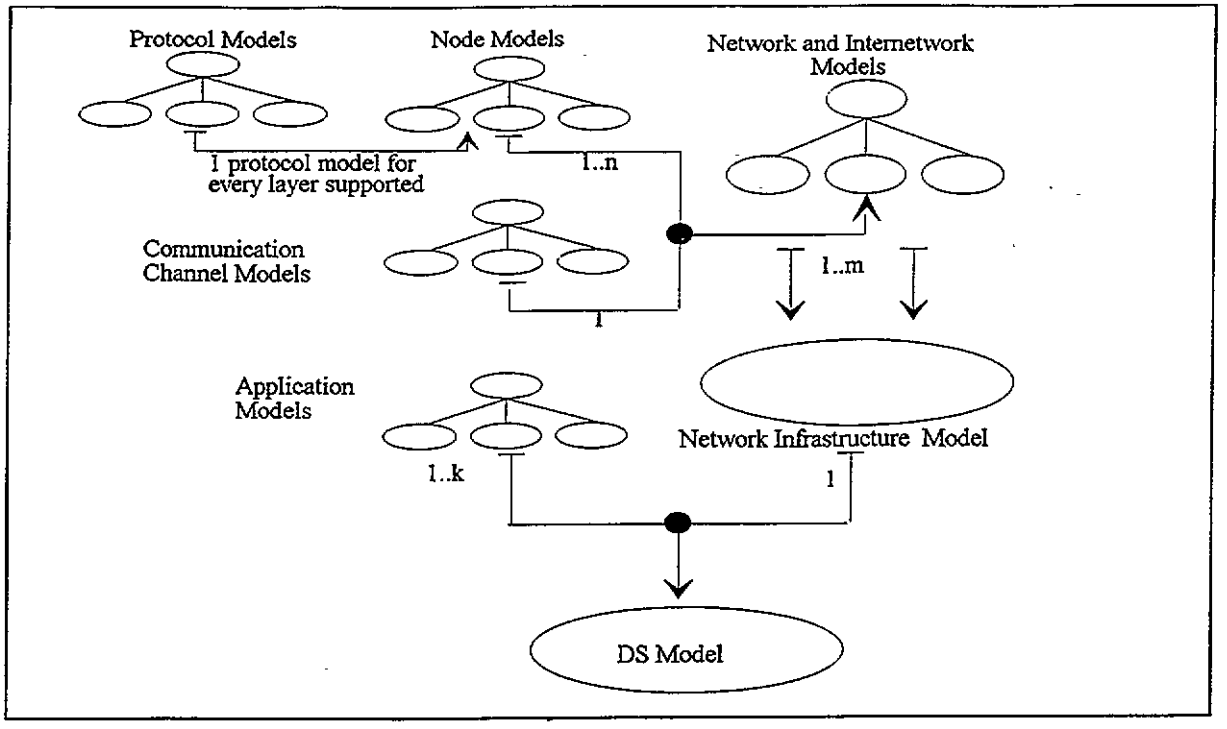

Figure 5. Abstract view of the proposed modelling schema.

infrastructure model and the distributed application models. The network infrastructure model consists of network and internetwork models, which are formed as a collection of node models (process or relay nodes), that in turn are composed of communication protocol models, and communication channel models. An abstract view of the proposed modelling schema is presented in Figure 5.

After the completion of the simulation process, simulation results are stored using the frame formalism in the fact base *Simulation Results* and are consequently processed by the Manager. Performance measurements are obtained for all resources, communication networks and distributed applications, as average response time and delay for all the front-end processes, total network throughput, resource and protocol utilisation.

6. Applicability and examples

To demonstrate IDIS functionality, a simple application case study is examined. This study concerns the distributed database used in the Admission Office of our Department in the University of Athens. The Admission Office premises are located in four different buildings, three in the University Campus (*Main Office, Department A* and *Department B*) and one at the Central University Building downtown (*Department C*). Two applications

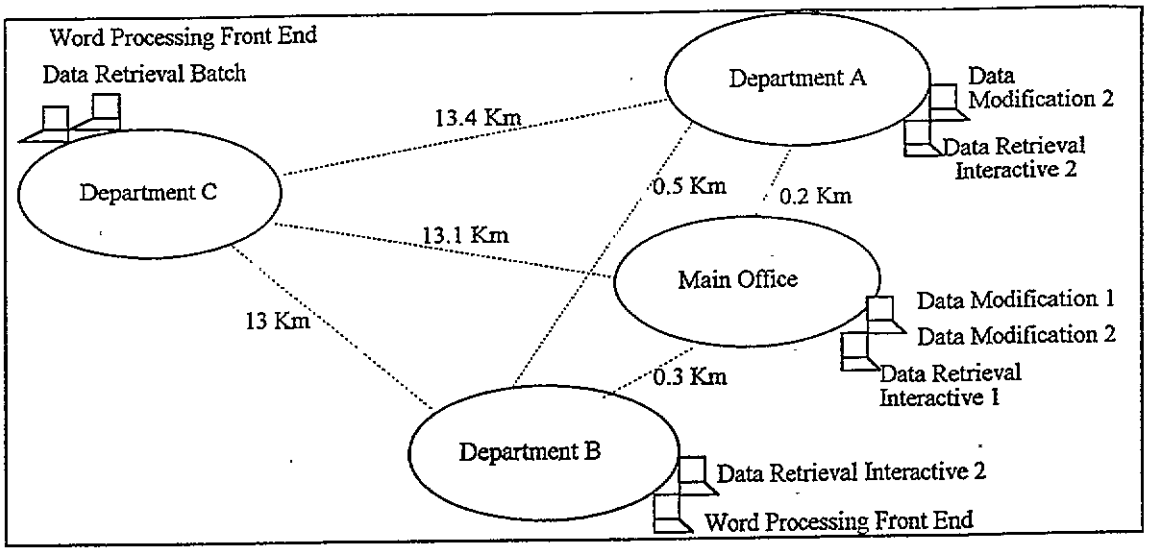

Figure 6. A simple distributed system example.

are examined. The first application is the Admission Office Database, which consists of two individual databases viewed as back-end processes, one for staff information and statistics and one for students. Both are updated and examined from different locations, as shown in Figure 6. The second application is a simple word-processing application. For all processes corresponding to each application their characteristics and functionality are described by the IDIS operator. For example, the *Data Modification 1* front-end process is invoked in *Main Office* by three persons from 9 a.m. until 6:15 p.m. with 15 min variation. Each time it is executed, the application code must be fetched from the File Server, and the database *DB1*, which contains staff information, is invoked. All actions performed during the execution of *Data Modification 1* are specified via *processing( )*, *request( )* and *diskIO( )* predicates.

The back-end process placement is accomplished by the ToDeM according to the algorithms described in Section 4.1. Only one possible LAN internetwork interconnects locations within the university campus. Both back-end processes, *DB1* and *DB2*, are placed within this LAN since *DB2* is accessed from locations within the campus and *DB1* is accessed mainly from the Main Office (for modification and retrieval) and partially from *Department C*. File Servers are placed in all locations except *Department B*, which uses the FS in *Department A*. When back-end process placement is completed, the internetwork type is determined and the User Knowledge predicates are updated.

An example of the architecture proposed by NeCoM for the location *Department A* is presented in Figure 7. Note that since there is a remote location connected with the Main Office through a WAN, a connection oriented protocol is chosen for the transport layer (TCP). The network protocol chosen for the LAN internetwork is IP, but the X.25 public network is used for the interconnection of the *Main Office* and *Department C*. For the LAN within location *Department A*, the Token Ring protocol is suggested since most jobs are interactive and a predetermined execution time limit is imposed. For the internetwork connecting locations within the University Campus, the CSMA/CD protocol is suggested, due to low traffic levels and the small distances between locations.

IDIS results are also applicable when designing distributed systems in the banking sector. IDIS was used to support our efforts towards designing and evaluating the new version of the DS operated by a medium size commercial bank to support its branches. Although for historical reasons most banking system implementations do not currently exploit general purpose DS techniques, all the requirements imposed are in areas for which formal distributed system design methods can be introduced.

The headquarters of the bank are located in Thessalonika. The bank also maintains 64 branches located in Thessalonika, Athens and ten other smaller cities in Greece. The bank supports one central database in its headquarters, where all transactions are recorded, and local databases in all branches. For the communication purposes, leased lines are used. Since back-end processes were strictly defined and located for protection reasons, IDIS was used to allocate file servers, define the network topology and evaluate the performance of the system. It also helped us considerably during the description of the user requirements, since it enabled us to estimate the exact amount of data processed and transferred within and between branches.

IDIS suggested that the HSTP protocol was the most suitable for the implementation of the transport layer and IP for the network layer. For the implementation of most of the local branch networks, the Token Ring protocol was chosen to ensure a predefined response time for all client transactions, although for some of the branches CSMA/CD was preferred due to low traffic. For the interconnection of all branches with the headquarters, IDIS determined the throughput necessary for the Greek

```
communicationElement(1,[DepartmentA],          /* Communication Element describing the protocol stack */
[peerCommunicationElement(1,[],[               /* proposed for location DepartmentA */
    application(1,[],[rpcapl(1,[0,2550,0,2550,blocking,request-reply,at-least-once,no,interactive],[])]),
    presentation(1,[],[rpcpres(1,[1,2550,0,2550],[])]),
    session(1,[],[rpcses(1,[0,2564,2550,14,client- server],[])]),
    transport(1,[],[transConOR(1,[tcp,0,2572,8 2572, ,bytes,go-back-n,3-way,2-way,yes,1]],[])])])]),
routingCommunicationElement(1,[],[
    network1(1,[],[netconless(1,[ip,0,2608,36,8192,255,'24$8',destributed,sortestPath],[])]),
    datalink(1,[],[llc(1,[0,2612,4,7,'PADs',go-back-n,2-way-handshake,2-way- handshake,3],[]),
            mac(1,[],[token-ring(1,[0,2612,26,4,3,2,4,'10msec',int],[])])]),
    physical(1,[twistedPair,baseband],[])])]).
relayProcessingNode(1, [DepartmentA, [3], 104850], []).   /* Relay Node used for interconnection*/
                                                          /* with the backbone internetwork */
processingNode(P1,[ DepartmentA, [FS], 66550,1], []).   /* Processing nodes needed in location DepartmentA */
processingNode(P2, [DepartmentA, [DB2], 45078,1], []).
processingNode(P3, [DepartmentA, [DataModification2], 14650,6], []).
processingNode(P4, [DepartmentA, [DataRetrievalInteractive], 35750,2], []).
storageDevice(D1, [DepartmentA,20000000,66500], []).   /* Storage Decive needed for the FS in DepartmentA */
```

Figure 7. NeCoM's proposed architecture for location Department A.

PTT. For all the branches the workstation-server model was suggested, although the bank was already using host RISC computers with terminals. The bank uses the TCP instead of the HSTP for commercial reasons.

The main contribution of IDIS was in the performance evaluation of the system, since it help us to determine the weak points of the previous version of the system and to ensure the response time expected for all client transactions. The average transaction response time was 20–25 s with the previous version of system, although with the new system this is expected to be reduced to 15–17 s. Finally, while the new version was being tested, IDIS also indicated that the processing power of the hardware supporting the central database was not enough in order to support all client transactions within the predefined response time. This was proved to be correct, forcing the bank to upgrade the obtained hardware.

7. Conclusions

Our main objective was to build a unified environment for both the design and evaluation of distributed systems. To achieve this, artificial intelligence and simulation methodologies were integrated in an IDIS framework. IDIS has been developed to assist DS designers during the construction of both experimental and commercial systems. It allows them to explore many different options in critical situations, and thus to increase the DS performance. IDIS is running in a SunOS environment. All ADT modules and SPGT are implemented using Sepia Prolog, and Model Libraries are implemented using the MODSIM II simulation language for SunOS. Future work will focus on the integration of new protocols parametrically described by IDIS operator and the development of a graphical user interface.

References

[1] Mills R and Skinner S 1993 COMNET III: The new enterprise-wide performance analysis tool *MASCOTS '93*, Simulation Series, vol 25, (1)

[2] Law A M and McComas M G 1994 Simulation software of communications networks: The state of the art *IEEE Communications Magazine*

[3] Vemuri V 1991 Simulation of a distributed processing system: a case study *Simulation Magazine*

[4] Jones S P W and Smythe C 1993 A generic framework for the simulation analysis of protocol layered communication systems *MASCOTS '93* Simulation Series, vol 25, no 1

[5] Kramer J, Magee J and Sloman M 1992 Configuring Distributed Systems *Proc 5th ACM SIGOPS Workshop on Models and Paradigms for Distributed Systems Structuring, Mont St. Michel, September 1992*

[6] Bagrodia R L and Shen C 1991 MIDAS: integrated design and simulation of distributed systems *IEEE Trans. on Software Engineering*, 17

[7] Shivaratri N G, Krueger P and Singhal M 1992 Load distributing for locally distributed systems *IEEE Computer Magazine* December 1992

[8] Ceri S and Tanca L 1990 Expert design of local area networks *IEEE Expert Magazine* October 1990

[9] Leung K S and Wong M H 1990 An expert-system shell using structured knowledge—an object-oriented approach *IEEE Computer Magazine* March 1990

[10] Sato A and Hamalainen R P 1990 Seteli: The Strategy Expert for Telecommunication Investments *IEEE Expert Magazine* October 1990

[11] Riveline C 1986 L'enseignement du dur et l'enseignement du mou *Gérer et comprendre* Décembre 1986, pp 42–5

[12] Nikolaidou M, Lelis D *et al* 1994 Intelligent design of distributed systems *Proc. 5th Int. Conf. on Database and Expert System Applications, September 1994*

[13] Stallings W 1989 *Local Networks: An Introduction* 3rd edn

[14] CACI Products Company 1993 *MODSIM II: the language for object-oriented programming* Reference Manual, January 1993