2nd Hellenic Conf. on AI, SETN-2002, 11-12 April 2002, Thessaloniki, Greece,
Proceedings, Companion Volume, pp. 511-522

Employing a Knowledge-Based Approach for the Configuration of Distributed Applications

M. Nikolaidou[1], D. Anagnostopoulos[2]

[1] Dept. of Informatics, University of Athens,
Panepistimiopolis, 15771 Athens, Greece
mara@di.uoa.gr

[2] Dept. of Geography, Harokopion University of Athens,
70 El. Venizelou Str, 17671, Athens, Greece
dimosthe@hua.gr

Abstract: Use of distributed systems is spreading and relevant applications become more demanding. To achieve the desirable level of integration of distributed system components, experience from different knowledge domains must be combined resulting in methods and techniques of increasing complexity. Expert Systems may contribute to resolving such problems, as most distributed system designers use empirical rules (rules of thumb) which can be represented and exploited by symbolic calculus. We present an Intelligent Distributed System Design tool, which facilitates the design and evaluation of distributed systems architectures making extensive use of artificial intelligence methodologies, mainly addressing the issue of configuring distributed applications.

Keywords: Knowledge Representation, Knowledge Exploitation, Distributed Application Configuration

1. Introduction

Distributed systems (DSs) can be described as sets of discrete components, such as applications, files, processing nodes and computer networks. These are bound together to interact and collaborate to achieve an overall objective. Applications operating in a distributed environment are generally called distributed applications. Most are based on the client-server model and its extensions, such as the two-tier, three-tier and multi-tier models discussed in [1]. 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.

The efficient configuration of DS environments is based on the successful combination of elementary components, also facing the internal complexity of these components. It requires the complete and accurate description of user specifications and the integration of knowledge from different areas. Among other factors, the large number of possible architectural solutions and the variety of distributed applications impose the development of software tools aiming at assisting experts during the construction, modification or performance evaluation of distributed systems ([3],[4]).

Traditional methods, such as simulation, have extensively been used for the design and performance evaluation of distributed systems and networks. Simulation tools often explore data and resource allocation problems using mathematical models, simulation techniques or, more often, a combination of both. Most tools aim at allowing experts to investigate the behaviour of predefined algorithms for the placement of resources and processes or estimating the performance characteristics of a given network architecture, performing a "what-if" analysis. They do not make suggestions for the design or redesign of the system architecture.

When designing complex systems, experts rely more on experience than on theory-based calculations ([5], [6]). Several approaches to solve application placement and network configuration problems appear in the literature. These approaches deal with subproblems of limited scope and, in most cases, are not applicable in practice. Instead, DS designers are driven by experimental rules of thumb that can easily be represented by rule-based systems. Moreover, the distributed system configuration problem requires solving several interrelated issues, that a heuristic approach can manage efficiently. Issues, as process and file allocation, proved to be NP-complete. Dealing with such problems requires methods that are more heuristic than algorithmic in nature.

Research in the expert system domain has often concentrated on the representation and manipulation of heuristic knowledge and its use in decision making. Artificial intelligent methods are widely applied for the effective exploration of *network configuration* problems ([7], [8]). Most are built to explore specific design issues and appear to have inherited the typical weaknesses of expert systems (inefficiency to exploit large knowledge bases, integration with other systems, etc). Even though, some tools include a special module for validating the proposed solution and guarantee that the generated architecture is correct in terms of compatibility, they fail to ensure that this is the most efficient solution, since they do not support performance evaluation of the proposed solutions. Discrete event simulation has proved to be an efficient aid for estimating performance characteristics of distributed systems and networks ([9], [10], [11]).

The research presented in this paper is oriented towards the construction of the *Intelligent Distributed System Design* tool, abbreviated to *IDIS*. IDIS makes extensive use of AI techniques and propose alternatives for the distributed system architecture according to the specifications provided by the user and the currently available technology. Its contribution involves the employment of knowledge-based techniques for the configuration of distributed applications and the description of problems encountered and the solutions proposed. A simulation module is incorporated into IDIS

environment, in order to facilitate the performance evaluation of the proposed distributed architecture and to guarantee the satisfaction of all user requirements.

The rest of the paper is organised as follows: In section 2 the distributed system configuration problem is discussed. In section 3 IDIS architecture is described. In section 4, we discuss knowledge representation issues in the context of distributed system design and the techniques adopted for reducing the complexity of the inferencing process. Conclusions reside in section 5.

2. Configuring Distributed System

Distributed systems are viewed as a combination of two discrete components: the distributed applications to be supported and the underlying network platform. Both application functionality and application requirements from the supporting network must be in-depth modelled and analysed for the construction of a distributed system ([13]).

The configuration of distributed systems is performed based on the client-server model, either two-tiered or multi-tiered. Systems based upon this architecture usually extend to multiple sites and use a variety of local and wide area networks. Hardware architecture is based to the workstation-server model: Users have their own workstation (diskless or not) for executing client processes. Server processes are executed on dedicated servers. All data used by applications are accessed through File Servers.

The network infrastructure consists of local networks interconnected via local and wide area internetworks. Networks and internetworks are represented through the corresponding protocol stacks, which are described according to the ISO/ OSI reference model. All protocols selected are de yure and de facto standards and considered to fulfill current and future communication needs.

Within IDIS framework, both distributed applications and network infrastructure entities are described by elementary components ([1]). The network infrastructure consists of *nodes*, either *processing* or *relay*, *storage devices* and *communication elements*. Communication elements represent networks and the protocol stacks supported by the corresponding nodes. For application description, two elementary components are introduced: *processes* (clients, servers) and *files*, accessed through File Servers. Users are described through *user profiles*. A typical DS architecture described in terms of the elementary components mentioned above is depicted in figure 1.

Distributed architecture proposals should be oriented towards:

- placement of server processes and data operating in the distributed environment to minimise network traffic and ensure the efficient operation of the DS
- design of the network infrastructure (network topology) to satisfy the requirements imposed by distributed applications

To effectively determine DS architecture, the complete and accurate description of the supported applications must be ensured. Furthermore, one should have the opportunity

to evaluate the performance of the proposed solutions and reconfigure the DS architecture if user requirements are not fulfilled.

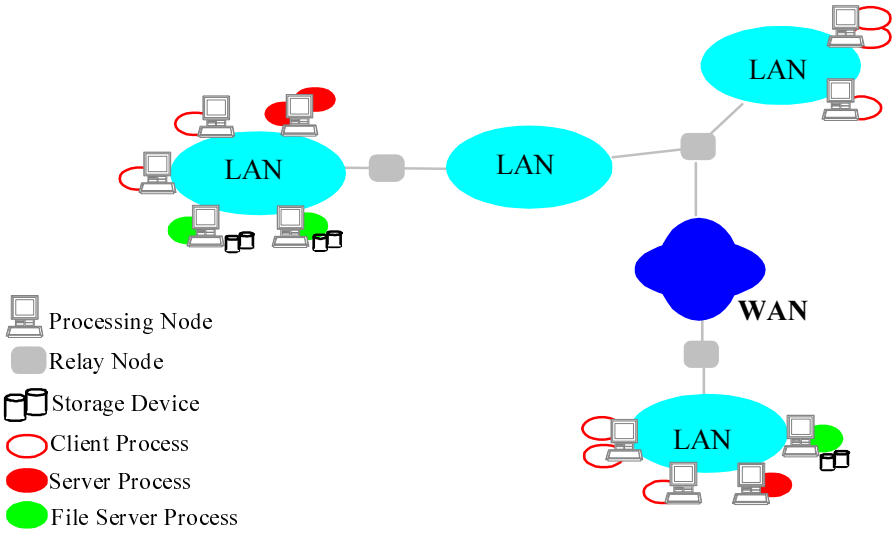

Figure 1. A Typical DS Configuration

IDIS provides a semi-automated environment, guiding the user throughout the configuration of distributed applications, which is accomplished in the following steps:

- functional topology definition
- logical topology definition
- physical topology definition

2.1. Functional Topology

Functional topology definition concerns the functionality of the system. Distributed applications are defined in terms of their operation and interaction with the network infrastructure. At this stage, the user defines applications without system interference, being responsible for the complete description of applications, while IDIS is responsible for testing the correctness of this description. Applications are described as sets of interacting processes activated by user profiles. More about the application representation scheme is included in [13]. The files used by all processes are also specified. There are two kind of files, data files and code files.

While interacting with the operator, access points of the DS called *locations*, distributed applications and their functionality and user profiles are specified. The operator also provides control information concerning specific conditions for the

performance evaluation of the proposed system. Definition of locations as well as specification of their size is performed with respect to the user's view. Locations are configurable and can be either different floors in a building, different buildings in the same area, different areas, or even a combination of the above. If, for example, we describe the sales database of a small firm, locations would most likely be different floors corresponding to different departments in the firm. If the information system of an industrial complex is described, locations could be floors in the corporate building, buildings in the corporate park or remote branch offices at different cities, as shown in figure 2. The *location* entity can be refined into more elementary ones, allowing the user to adjust the description of the system according to the application scale. This also applies to other main entities, such as networks and internetworks, which are defined during physical topology construction.

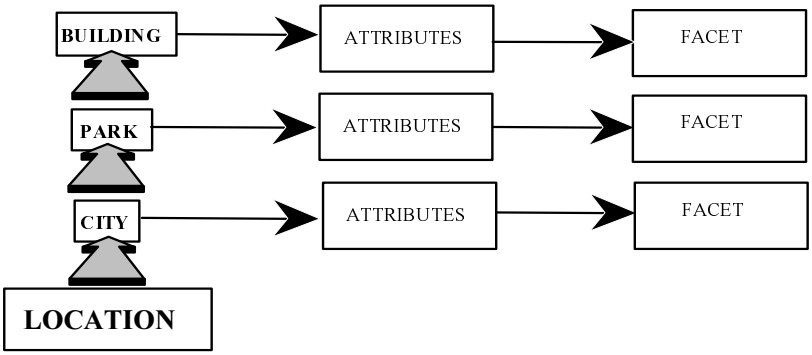

Figure 2. Progressive Refinement of Location Definition

2.2. Logical Topology

Logical topology definition ensures providing the required functionality. Applications requirements from the network infrastructure are described by parameters, such as the necessary throughput, the application type (batch or interactive) etc. They are estimated from their functionality description provided by the operator.

Allocation of processes and files is performed aiming at a. fulfilling application requirements and b. minimising configuration cost. Minimising network traffic especially on WAN connections and balancing the load is also taken into account. While locating processes and files, different replication scenarios can be applied. When dealing with the file allocation problem (FAP), i.e. the optimal allocation of file copies, one must usually minimise the function describing the overall communication cost for file access. This, as proved in [12], is considered to be NP-Complete. The same applies for program allocation as well. For the solution of this kind of problems, heuristic or similar techniques are introduced, ensuring that even if the optimal solution is not reached, one very close to it will be found.

2.3. Physical Topology

At the stage of physical topology definition, the network topology is designed. The network infrastructure consists of local networks interconnected via local and wide area internetworks. Since the performance of distributed applications depends critically on the performance of the network infrastructure, special attention is given to the way locations are interconnected. The "best" solution to this problem is to interconnect each location with all others communicating with it. Unfortunately this solution is to costly to be supported. Graph processing techniques and heuristics are introduced to indicate an "acceptable solutions" satisfying application requirements. Networks, internetworks and resources characteristics must also be defined. IDIS provides alternatives for process and file placement, network topology and network configuration to ensure the desired performance, but does not indicate commercial solutions

3. IDIS Architecture

IDIS is a knowledge-based system constructed according to the blackboard architecture. Its knowledge base contains information concerning the supported protocols and resources along with selection and combination rules and rules for process and data placement. The blackboard model allows the structuration of the knowledge base and the acceleration of the inference engine by dividing it into independent subunits, called knowledge sources, that exchange information through the working memory, referenced as blackboard. The use of blackboard architecture facilitated the efficient incorporation of the simulation environment into IDIS framework, as it permits the integration of subunits with different internal architecture, provided that they are able to communicate using information in a predefined format. The blackboard architecture is also used for the internal structuration of all subunits. As Prolog programming language was used for implementation purposes, the input and output of subunits are Prolog clauses. The knowledge sources forming IDIS inference engine are supervised and directed by a control subunit, the Manager, as shown in figure 3.

The functionality of main IDIS subunits is analytically presented in [14]. The *User Interface Module (UsIM)* is responsible for the definition of the functional topology in cooperation with the operator. UsiM is a graphical environment implemented using Java Programming language. UsIM is also responsible to interact with the operator during distributed application configuration and evaluation. The *Topology Design Module (ToDeM)* is responsible for the definition of the logical topology, while the *Network Configuration Module (NeCoM)* is responsible for the definition of the physical topology. ToDeM functionality is defined by a set of rules consisting of formal descriptions of experimental, mathematical and empirical techniques for data and process placement, topology design and application requirement computation. NeCoM functionality is defined by a set of rules concerning network design and configuration. This knowledge is permanently stored in the Knowledge Base and can be subjected to update by IDIS proper mechanisms.

The *Performance Evaluation Module (PeM)* is responsible for evaluating alternatives solutions using discrete event simulation. MODSIM simulation language ([9]) was used for simulation purposes. The simulation program is automatically constructed based on object libraries. When its execution is completed, performance evaluation results are stored in the corresponding knowledge base. The DS configuration constructed fulfils all the desired performance characteristics (for example avg. or min. application response time), while its cost is also taken into account.

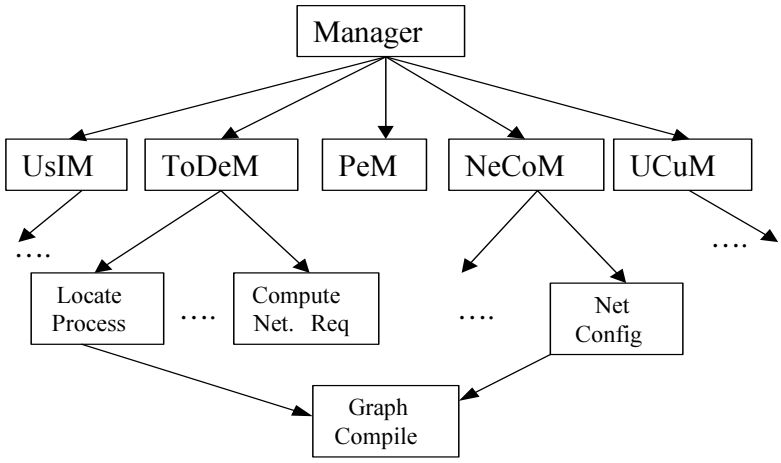

Figure 3. IDIS Architecture

The *User Customisation Module (UCuM)* facilitates the enrichment of IDIS knowledge base. It allows the extension of the models used for application description, the addition or re-organisation of process and file allocation policies and the modification of network configuration policies. Extending IDIS knowledge base is not trivial, since it may effect reasoning ([15]). For example, extension of application models may also affect the algorithm computing network requirements. In order to avoid knowledge inconsistency, the operator's ability to add or modify knowledge is restricted. Thus, UCuM functionality is constructed by meta-rules ensuring accuracy and consistency of the added knowledge. For this purpose the structure of the added knowledge is predefined.

Tasks accomplished by IDIS cannot be achieved by algorithms within acceptable processing time or cost. Only partial answers may be given within an acceptable time frame. Since the problem IDIS copes with is semi-structural, it can be resolved with much simplicity when, at *critical points* of the inferencing process, an experienced user interferes and reorients/guides the system. Thus, co-operation of an DS configuration expert with IDIS is essential.

Critical points (Process Break Points) are the ones where IDIS must make a choice according to specified criteria. The time required for the exhaustive combinatorial examination of all alternatives can be drastically reduced by the operator. During process placement, for instance, the number of alternative solutions can be reduced if the operator excludes some remote locations from the system initial choices. The term

intervention denotes the provisional interruption of IDIS operation, the interactive transformation of data by the operator and the reactivation of IDIS inference engine. IDIS resumes its operation using the modified data. These interventions, even though desirable, can also be fatal. Taking into account that the selection of each successive task depends on the form of the data processed, data modification affects the control itself. Allowing continuous and insufficiently controlled external data modification could seriously disturb the system. To ensure IDIS efficient operation, Process Break Points are strictly defined.

4. Knowledge Representation

The three classic formal descriptions of knowledge, i.e. frames, rules and semantic networks, are used for knowledge representation. Frame-based reasoning is used for entity generation, while rule-based reasoning is mainly used for algorithm representation.

4.1 Frames

As frame-based reasoning simplifies configuration procedures, frames are used to represent all entities describing DS components. Inheritance properties are extensively supported. The frame structure introduced and examples of DS entity representation are presented in figure 4.

```
/* Frame structure */
frame_name(ID, Is_A_List, Slot_List, Has_A_List)
where Slot_List is Local_Slots ∪ Inherit_Slots

/* rule structure for property initialisation */
find_frame_name (slot, value)

/* rule structure for entity initialisation */
find_frame_name_has_a(Frame)

/* sample frames representing Distributed Applications */
location(ID, [], [{Other Loc, Distance}], [Refined_Locations]).
application(ID, [], [], [{process_list}, {file_list}]
process(ID, [process], [kind, response_time], [{interface,
{operation_scenario}]).
operation_scenario(ID, [ ], [Interface_name], [Process,
Action_List])
request(ID, [ ], [Interface_name, Server, Request_size,
Amount_size], [ ]).

/* sample frames representing Network Infrastructure */
network(ID, [ ], [LocationList, type], [CommunicationElement]).
communicationElement(ID, [ ], [LocationList], [PeerComEl,
RoutingComEl]).
processingNode(ID, [ ], [Location, ProcessList, ProcessingPower,
Number], [ ]).
```

Figure 4: DS Entity Representation

During applications description, UsIM is responsible not only for ensuring that the information provided by the user conforms with the predefined structure and form, but also for checking on potential contradictions and omissions (knowledge acquisition control).

Normally, the operator does not change the chains of successive attributes which describe a frame. However, if required, there is a mechanism which permits this type of modification. Through UCuM, IDIS operator can adapt the description of the entities to her/his model: attributes can be either added, modified or eliminated. These changes result in updating the dictionary. Any modification in entity description is followed by knowledge validation, which is performed by daemons before adding it to the dictionary ([16]). Daemons can also be modified, but this "privilege" is granted only to IDIS developers. The function of daemons is founded in the use of rules, which are fired by changes in the frame structure.

Rules of knowledge integrity prohibit from significant changes in the continent of the Knowledge Base. As a result, knowledge acquisition, validation and verification procedures do not overcharge the system with exhaustive time-consuming checks. On the other hand, experimental use of IDIS has shown that users usually wish for insignificant modifications to the description of entities. As the domain of distributed systems permits a quite complete and objective description of the knowledge concerning DS entities, a detailed test of knowledge acquisition, validation and verification made by complicated mechanisms, which permit radical changes of the knowledge base status, is considered as unnecessary.

4.2 Rules

During IDIS operation, the operator instantiates frames corresponding to DS entities, which instantiate other frames, etc. A subunit of the inference engine is activated, when all necessary frames are instantiated and the corresponding set of rules (bank of clauses) is loaded. The subunit checks on the compatibility of the instances, e.g. it checks if certain values of instances exclude/impose some others and if the excluded/imposed ones do not/do appear among the instances. The subunit also provides a relatively "good" choice in case a frame can be instantiated in more than one ways. If some combination of instances is considered to be better than the combinations resulting from the user choices, the subunit updates a temporary bank of clauses in which alternative scenarios are maintained. The system will consider these scenarios in case of inability to provide a solution with respect to user choices and real world constraints. The alternative scenarios are considered as part of the intermediate results, which the system must keep in order to make new attempts in case of a failure. They are not exactly what the user has asked for, but they are more "correct", in a technical sense, so that they are considered as "legitimate by-passes" to intractable difficulties. For example, during logical topology description, the user may choose to place a server process replica in a location with many interactive applications to increase their performance, and avoid placing it in the nearby location to increase the overall system performance. Even if the user insists, IDIS must keep the alternative scenario. The operation of rules is depicted in figure 5.

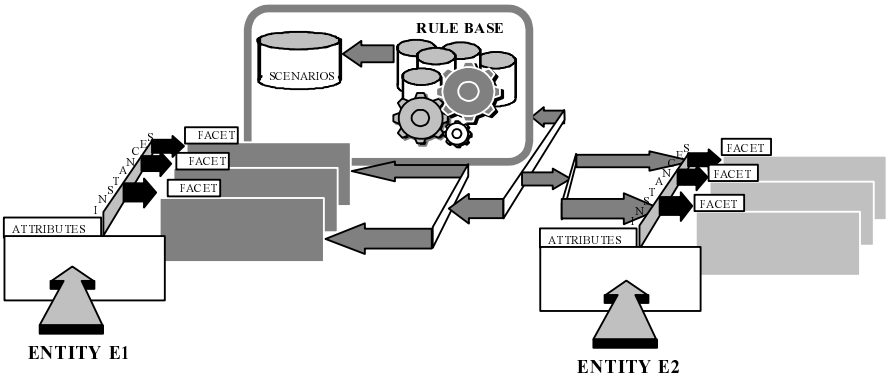

Figure 5. Rule Functionality

Rule-base reasoning is used for algorithm representation. For example, during process allocation different allocation scenarios must be represented. The most popular algorithm for server placement is the one based on the avoidance of unnecessary data transfer between WAN connections with no data replication support. The algorithm does not support optimal performance solutions, since it only focuses on WAN traffic and searches for a "relative good", cost effective and simple (no data replication) solution. The algorithm is executed in independent steps presented in the following:

Step 1 For each server process, find the client process sets that use shareable data.

Step 2 For each set, construct a different server process replica. To place the replica:

I. Find the sets of locations that can be interconnected via a LAN. (If a set contains only one location, the location can not be interconnected via a LAN).

II. Find the average network throughput caused by data transfer, involving clients using shareable data, for each location.

III. Find the average network throughput caused in all possible LAN internetworks.

IV. Place the Server replica in the LAN internetwork that causes the maximum network load.

Step 3 For each LAN internetwork with one or more server replicas, place a server replica in the location that causes the maximum load.

After the successful execution of each step, intermediate knowledge is kept to effectively support backtracking. Alternative algorithms supporting different data replication schemes and LAN traffic minimisation are also supported. Each supported algorithm has an *Activation_Factor,* indicating activation order. Most simple algorithms, such as the one aforementioned, are first applied. If the proposed solution does not satisfy application requirements, more complex algorithms are tested usually resulting in more costly solutions. *Activation_Factor* of each algorithm may be altered by the operator.

4.3 Structural Knowledge Factorisation

Both locations used for distributed application description (functional topology) as well as networks and internetworks used to describe the network infrastructure (physical topology) are internally represented by semantic networks, which in turn are represented as graphs. All locations defined by the user form a semantic network containing information about the distance between locations. In the corresponding graph, locations are viewed as nodes. An edge exists between two nodes if the distance between the corresponding locations is less than 2 Km, meaning that they can be interconnected within the same LAN. This information is used during server process placement and network topology definition, where alternative scenarios for location interconnection are examined. In this graph, all possible LAN internetworks are represented as complete subgraphs (every node communicates with all others).

Graph complexity makes its manipulation laborious. To overcome this, information contained in the graph is *factorised*. The objective of factorisation is to facilitate the subunits of the inference engine by indicating where to look for specific information, that is, to direct them to this part of the Knowledge Base where relevant information is maintained. Factorisation of the knowledge fragments with respect to their meaning, however advantageous, is rather difficult to achieve without the drawbacks that exhaustive research imposes: the inference engine does not know if a fragment of knowledge contains useful information before it examines it. The factorisation that has been made is founded on structural criteria. To effectuate structural factorisation, the corresponding graph is fragmented. The advantage of factorisation is that, in case of a modification of the semantic network, the inference engine does not need to repeat the whole process. Such a modification requires a minor restructure of the Knowledge Base for the inference engine: a modification of the semantic network implies the addition or the elimination of vertices or/end edges. Additions/eliminations of these elements create/eliminate loops which could create/unify/eliminate strongly compound components or complete subgraphs. Taking into account that users usually modify the semantic network corresponding to the distributed system they wish to design (e.g. addition of a new location), keeping changes as local as possible becomes vital.

The same techniques are also applied to the construction of simultaneous process sets, which are used to estimate maximum application requirements from network resources, and also during process placement.

5. Conclusions

IDIS has been developed to assist distributed system developers during the design of a new system or during the improvement of an existing one and allows the exploration of different options to increase distributed system performance. To support the design of real scale distributed systems extending to multiple locations, the acquisition of a dynamically increasing Knowledge Base is facilitated. Grouping the knowledge provided by the operator according to specified criteria (e.g. find the groups of simultaneous processes) and efficiently estimating the results of complex computations (e.g. compute maximum requested throughput for each network) were thus critical.

Knowledge factorisation techniques were introduced to increase inference engine performance and proved to be extremely useful, especially in avoiding inference engine saturation due to a great number of backtracking points, as in graph compilation.

References

1. Shedletsky J., Rofrano J.: Application Reference Designs for Distributed Systems. In IBM System Journal **32**. IBM (1993)
2. Coulouris G.F., Dollimore J., Kindberg T.: Distributed Systems - Concepts and Design, Third Edition. Addison Wesley Publishing Company (2000)
3. Coutts I.A., Edwards J.M.: Model-Driven Distributed Systems. IEEE Concurrency **5**. IEEE Computer Press (1997)
4. Savino-Vazquez N.N., Anciano-Martin JL, et. al.: Predicting the behavior of three-tiered applications: dealing with distributed-object technology and databases. Performance Evaluation **39**. Elsevier Press (2000)
5. Juengst W.E, Heinrich M.: Using Resource Balancing to Configure Modular Systems. IEEE Intelligent Systems **1**. IEEE Computer Press (1998)
6. Fleischanderl G., Friedrich G.F., et. al.: Configuring Large Systems Using Generative Constraint Satisfaction. IEEE Intelligent Systems **1**. IEEE Computer Press (1998)
7. Ceri S., Tanca L.: Expert Design of Local Area Networks. IEEE Expert **2**. IEEE Computer Press (1990)
8. Dutta A., Mitra S.: Integrating Heuristic Knowledge and Optimization Models for Communication-Network Design. IEEE Transactions on Knowledge and Data Engineering **5**. IEEE Computer Society (1993)
9. Khoroshevsky V. D.: Modelling of Large-scale Distributed Computer Systems. In Proceedings of IMACS World Congress, Vol. 6. IMACS (1999)
10. Ramesh S. R.: An Object-Oriented Modeling Framework for an Enterprise-Wide Distributed Computer System". In Proceeding of the Americas Conference on Information Systems. Association for Information Systems (1998)
11. Matsushita M., Ashita M., et. al.: Distributed Process Management System based on Object-Centred Process Modeling. Lecture Notes on Computer Science 0302-9743. Springer Verlag (1998).
12. Eswaran K.P.: Placement of Records in a File and File Allocation in a Computer Network. In Information Processing '74. North-Holland Publishing Company (1974)
13. Nikolaidou M., Anagnostopoulos D.: An Application-Oriented Approach for Distributed System Modelling. In Proceedings of IEEE ICDCS 2001. IEEE Computer Press (2001)
14. Nikolaidou M., Lelis D., et. al.: A Discipline Approach towards the Design of Distributed Systems, Distributed System Engineering Journal **2**, IEE Society Press (1995)
15. Lukaszewics W., Madalinska-Bugaj E.: Lazy Knowledge Base Update. In IEA/AEI 2001. Lecture Notes on Artificial Intelligence , Vol. 2070. Springer Verlag (2001)
16. Debenham J.: Knowledge Base Maintenance through Knowledge Representation. In DEXA 2001. Lecture Notes on Computer Science, Vol. 2113. Spinger Verlag(2001)