AI Applications in Distributed System Design

Introduction

Distributed systems, such as those which contain heterogeneous operating systems, are crucial to the design and development of new and innovative products for the future. Differences in operating systems will continue to persist because of the variety of hardware in the marketplace, each with software that provides a vendor with a product having a customized or competitive edge over the other [1]. The design and implementation of such systems pose many problems. As is pointed out in [2], even in homogeneous systems, operating systems of the present generation provide little or inadequate support for local area networking and distributed processing. Non-uniformity in the characteristics of current operating systems has posed additional complexity in the development of heterogeneous local area networks. This has led to some researchers of distributed systems incorporating AI techniques into the designs of various aspects of these systems. Notable among these efforts is the knowledge-based distributed operating system work reported in [2]. Here, a revolutionary approach is proposed to design distributed systems using object-oriented techniques that support local area networking and distributed processing. AI techniques are also used in network management. For example, in [3] it is reported that expert systems have been developed to fight congestion in networks.

This paper concentrates on how AI techniques can be applied to enhance the design of the front-end subsystem of each host in a distributed environment. The functions of the front-end subsystem that will be addressed in this paper are the following:

- user interface management
- interconnection in the case of heterogeneous systems
- security

Each of the above functions is essential for the successful and useful operation of a distributed system.

The user interface is that part of the system that interacts directly with the user. The purpose of modern user interfaces is to ease the burden placed on the user as much as possible. This is realized by providing facilities such as browsing, command formulation, and automatic planning. Browsing helps users to locate objects. Command formulation helps users to formulate commands that are syntactically and semantically correct. Automatic planning hides the complexity of interacting with multiple sources of information from the user. Later in this paper, we will describe the issues that relate to intelligent user interfaces for distributed systems.

A special case of automatic planning is the interconnection of heterogeneous systems (i.e., a distributed system that consists of different operating systems). Interconnection is one of the major issues that need to be addressed in the design of heterogeneous systems. The current trends in interconnection as well as possible solutions to the problems encountered using AI techniques will be outlined later.

Security is another function for which the front-end subsystem should be responsible. Enforcement of security within each host could be part of the corresponding kernel's functions. However, it would not be expedient to incorporate security functions of a more general nature into the kernel. Security in distributed systems has received much attention lately due to the increasing number of applications of these systems in the military, process control, business, and financial areas, among others. We will discuss the security issues in distributed systems and the problems with the current approaches, and we will propose solutions using AI techniques.

Traditional approaches to the design of user interfaces, interconnection of heterogeneous systems, and security have been beset by problems. In this paper, we contend that an alternative approach to solving these problems is needed if companies are to take advantage of the market opportunities that will become available in distributed systems.
User Interfaces

Current Trends

Operating systems have traditionally used key-word-oriented command languages. Using these languages, the user formulates commands with the help of a text editor. These languages suffer two major problems:

- Users must learn and remember the names of various files and objects which are named in commands.
- Users must learn and remember the exact language syntax.

A third problem occurs in a distributed system with multiple operating environments: users must explicitly direct requests to specific operating systems, and explicitly coordinate and merge results from multiple operating systems.

Modern user interfaces are attacking all of these problems in the following ways:

- **Browsing** is used to help users locate files and objects.
- **Command formulation facilities** help users formulate commands that are syntactically and semantically correct.
- **Automatic planning** hides the complexity of interacting with multiple systems from the user.

**Browsing**

A visual approach to browsing through heterogeneous databases, files, and directories is one way for users to integrate information from the various systems. Users perform four basic operations when browsing:

- **structuring**: choosing a structure of the objects to be examined
- **filtering**: selecting instances of the objects to be examined
- **panning**: examining neighboring object instances
- **zooming**: determining the level of the detail for examining object instances [4]

**Structuring**

Knowledge acquisition and classification techniques are needed to help structure objects from different systems. Without training, humans generally do a mediocre job of extracting semantics, building cross references, and establishing the structure for diverse objects. Object-oriented approaches also appear promising as an approach for representing and manipulating objects. However, object-oriented approaches by themselves do not solve the problem of determining what the relationships among objects are. Knowledge acquisition techniques are needed to help humans establish the structure of objects. Clustering algorithms are useful for establishing classifications. Automated classification techniques can be used to identify subject classes to which a new object belongs.

**Filtering**

The time and effort expended searching can be dramatically decreased by restricting the search space to a character-ization of the class of relevant objects. Query languages can be used to project and restrict irrelevant data objects, leaving only potentially interesting data objects available for browsing. In order for users to filter large search spaces, they must be able to formulate commands that trim the search space.

For example, a user can view the thinned search space by specifying a pattern. A pattern is a form of template with zero, one, or more conditions. Figure 1 illustrates a pattern consisting of two conditions: the attribute for COURSE Name must be “Databases”, and the attribute for COURSE Number must be less than 5,000. An instance of an entity must satisfy all conditions in the window if it is to be displayed to the user.

**Panning**

Often the number of objects, even in a filtered search space, is too large to display on a terminal screen. Panning techniques, including paging and both horizontal and vertical scrolling, permit users to scan large numbers of objects.

The user views the entity instances of the thinned search space as a series of nested documents. The entity instances are displayed in order sequence as a nested document (Figure 2). The user may page forward and backward between documents. By choosing a paging increment, the user may jump forward or backward a specified number of documents. The user may also display the first or last document and scroll within documents.

**Zooming**

Users navigate through a structure of objects by zooming. Zooming can be used to change the level of abstraction of objects being displayed. "Zooming in" causes more detail to be displayed; "zooming out" causes less detail to be displayed. Opening and closing windows is a type of zooming in which a highly abstract object (window name) and a more concrete object (contents of the opened window) are interchanged. Zooming can also be used to change context: users can zoom to a related object or class of objects. Zooming is the user interface equivalent of the programming language command "go to".

For example, each entity instance corresponds to a window in the form of a template that may be either opened or closed. An open window displays all the attribute values of an entity instance. A closed window displays only the name.

![COURSE](image)

**Fig. 1. Sample pattern.**

**Table:**

<table>
<thead>
<tr>
<th>Number:</th>
<th>&lt;5,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>Databases</td>
</tr>
</tbody>
</table>

November 1988
IEEE Network
of the entity instance. Users may open or close any window, depending upon whether they wish the attribute values of instances of that entity type to be visible when the selected entities are displayed. Opening and closing windows allows the user to control the level of detail in which the results are viewed. Figure 3a illustrates a form template with the windows for SECTION closed. Figure 3b illustrates the result of opening the SECTION windows.

**Command Formulation**

Tools are needed to assist users in formulating operating system commands. While the browsing facility can be used to identify useful functions and data, the user must formulate syntactically and semantically meaningful commands himself. Facilities such as the following make the task of formulating commands easier for the user:

- Display a syntax diagram (a graph whose nodes represent key words and options of the command language) to guide the user in formulating syntactically valid requests [5].

- Detect interactive command syntax errors, present the user with the valid options for correcting each syntax error, and correct the error, depending upon the option selected by the user [6].

- Help the user construct valid commands by asking questions tailored to the user's interests and background, and formulate commands on behalf of the user.

- Paraphrase user-formatted commands into natural language for review by the user. If the user does not agree with the natural language paraphrase of the command, the command is likely to be semantically incorrect and can be modified by the user.

- Accept database requests entered via the keyboard as natural language text.

**Automatic Planning**

Users of distributed information systems often need to access multiple sources of information and software programs in the course of performing a single practical task. For example, a bank loan officer may need to access credit records, automobile book values, and amortization software to determine whether to grant a car loan. The use of multiple systems can burden users with unnecessary involvement in the mechanical details of obtaining and combining intermediate results. A means is needed whereby a person could access diverse information sources and software functions without being distracted from the task at hand by these decisions and details. A way to meet this need is to provide a user-system interface that allows a user to use diverse resources as if they were a single, virtual information system. An algorithm is needed that automatically creates a distributed execution strategy that selects the resources needed to respond to the request for information stated in system independent terms. We have found a graph-oriented technique for representing resource capabilities [8]. Used in conjunction with unification and heuristic search algorithms, it provides an approach to automatically selecting and invoking the appropriate resources to solve a user's problem. A key to this approach is the representation of user request and resource capabilities in a uniform way. Our implementation shows that the approach can formulate a set of server invocations in a tractable amount of time. Included as a component of a larger user interface [9], this approach can yield the benefits of a uniform interface to multiple heterogeneous systems.

**Interconnecting Heterogeneous Systems**

**Current Trends**

The previous section discussed issues relating to intelligent user interfaces. More specifically, the issues relating to browsing, command formulation, and automatic planning

---

**Fig. 2. Nested document.**
were discussed. A special case of automatic planning occurs when the user needs to invoke functions provided in multiple operating systems in a distributed environment. In this section, we will discuss the current trends in interconnecting heterogeneous systems and propose solutions (using AI techniques) to some of the problems. Some of these solutions have also been given in [1].

Interconnection was identified as one of the major issues that need to be resolved in the design of heterogeneous systems at the 1985 ACM SIGOPS Workshop in Eastbound, Washington, which focused on accommodating heterogeneity [10]. Interconnection deals with the selection of appropriate communication primitives and handling different data representations.

In the case of a homogeneous network, the various systems are usually interconnected by a separate layer, often called the connection layer [11], which is implemented as a user process. This layer is at the level of the application layer in the ISO model. Such a connection layer can be extended to handle the different data representations and commands to deal with heterogeneous systems. A product that is based on the connection layer paradigm that interconnects heterogeneous systems is the Network File System (NFS). This system provides transparent network file system services. It originally ran on SUN Microsystems’ workstations and has now been adapted to run on various machines.

Having the connection layer perform translations between the data representations and commands avoids the necessity of making changes to the Host Operating Systems (HOSs). Furthermore, a user needs to be familiar with only the HOS that he uses [12]. This is realistic, since it is unreasonable to expect even experienced programmers to know more than a couple of different operating systems. However, there are some problems with such an approach (see Figure 4).

We will illustrate the sequence of events usually performed by a nonintelligent connection layer when users request to access network resources.

First, the user issues requests to access local as well as remote resources in a format compatible with the HOS to which the user has an interface. Although this is a straightforward procedure, the user is required to have some knowledge of the HOS.

The connection layer then translates remote requests into a format compatible with the target operating system. Currently, ad hoc techniques are used to accomplish this. Furthermore, there may be requests that are not directly translatable, and consequently users may not have full access to exiting remote services.

Next, the translated requests are carried out in the target system. This again is a straightforward procedure.

Finally, the responses generated in the target system have to be translated into a format that can be understood by the requesting user. Here again, ad hoc techniques are now used to perform these translations.

These events are illustrated in Figure 5.

We will now describe how AI techniques may provide a more user-friendly environment. The solution that we propose is to extend Billmer and Carifo’s techniques [13] to teaching operating systems for naive users to a distributed system environment. In [13], two approaches are proposed for teaching operating systems to naive users:

- In the short-term approach, users of one operating system use a second operating system with the aid of an expert system, without actually having to learn the second system.
- In the long-term approach, the users learn only a generic operating system. With this knowledge, they are able to use many other operating systems with the help of an expert system.

The next two subsections will extend the approach in [13] to a distributed system. Note that approaches similar to those proposed in [13] have also been proposed in [14] and [15] to teach operating systems to naive users.

**Augmenting the Connection Layer with a Knowledge-Based System**

The short-term approach that must be taken to solve the current problems faced in the design of connection layers is to augment a connection layer with a knowledge base and an inference engine (see Figure 6) which will act as a front-end to the connection layer. This front-end will also encompass the user interface and will perform the following functions:

- Accept user requests specified in a format compatible with the HOS
- Translate these requests, if necessary, into a format compatible with the remote HOS

**Fig. 3. Opening and closing windows.**
Fig. 4. Interconnection.

Fig. 5. Processing user requests.

Note: Numbers in circles indicate the order in which the operations are performed.
However, in any particular translation process, only two operating systems will be involved. This drawback will be rectified in the long-term approach to be described in the next subsection.

In designing this front-end, the key issues that need to be addressed are:

- Knowledge representation mechanisms for the translation rules
- Inference strategies to efficiently process the knowledge base

As this method deals with only two operating systems during a translation process, a rule-based system that uses a forward-chaining inference mechanism will suffice. This inference engine will act on the rules of the knowledge base and perform the following functions in addition to the translations:

- Produce advisory messages to the user
- Interactively help the user to formulate requests
- Suggest approximate translations in cases where requests are not directly translatable

Therefore, our short-term approach described here not only solves some of the problems faced by current connection layer designs, but is more user-friendly than the current connection layer user interfaces.

**Generic Operating System**

Although the short-term approach described in the previous section enhances the functions of the connection layer, it does not gracefully handle cases where users are forced to interface with many operating systems. Thus, the users are still expected to be somewhat familiar with all the different operating systems. This is obviously unreasonable. Therefore, the longer-term solution described in this section is aimed at rectifying this problem. It does so by designing a conceptual, virtual operating system that creates the illusion to all users of the network that they are only using one operating system. However, in order to use this conceptual operating system, a user interface should be designed in which users can formulate their requests.

The approach to be taken here is as follows: As in the short-term approach, the connection layer will be augmented by a knowledge base and an inference engine (see Figure 7). The inference engine will apply the rules in the knowledge base to develop a distributed execution plan involving translation of both local and remote HOS requests.

The issues that have to be addressed in designing this system include the following:

- How to represent the network resources in the conceptual operating system
- What type of representation is used for the translation rules

---

*Note: Numbers in circles indicate the order in which the operations are performed.*

**Fig. 6. Knowledge-based interconnection.**
What search strategies are used
What heuristics should be used to develop a distributed execution plan
In the case of accessing remote requests, whether all the translations should be performed at the requesting user's site, or whether the request should be translated to some intermediate format at the requesting user's site and then sent to the remote site for further translations.

We will now discuss the possible approaches of addressing these issues.

One needs to first examine whether any of the existing operating systems would serve as the conceptual operating system. The UNIX™ operating system was designed to be just such a general purpose operating system. However, investigation is needed as to whether the UNIX operating system would provide a uniform and coherent view to all users, and whether UNIX's resources can be mapped into the resources of most existing operating systems (and vice versa).

A particular model that we recommend to be used for the conceptual operating system is the object model. This is because any resource can be modeled as an object. Objects will include traditional operating system resources such as files, devices, terminals, processes, and directories. Operations on these resources (e.g., reading from files) can be translated into operations on the data types to which the corresponding objects belong.

Two types of rules can be identified in order to successfully translate requests in the conceptual operating system format into a specific operating system format (or vice versa). One type transforms a user request into a distributed execution strategy. The other type transforms a request within a distributed execution strategy into a request to a specific operating system. All rules do not have to reside in all hosts (i.e., the knowledge base could be distributed).

The automatic planning algorithm of [8] is successful because it uses the same level of structure (a semantic, net-like representation) to represent both user requests and resources. Thus, we recommend that both user requests and operating system resources be represented with the same general structures (semantic nets and object-oriented techniques are good candidates). By also representing the translation rules using the same structure, a general-purpose mechanism can be used for translation and planning.

Security in Distributed Systems

Current Trends

We have described how AI techniques may be used to provide more useful interfaces to the users of distributed systems.

Fig. 7. Generic operating system.

UNIX™ is a trademark of AT&T Bell Laboratories

November 1988
IEEE Network
systems. Another important function we feel that a front-end subsystem must provide is to ensure that users have proper access to the information that they request. This information usually resides in files or in databases. Security in distributed systems has recently received much attention. This is because distributed systems are being developed for an increasing number of applications in the military, process control, business, and financial areas, among others. Many of these applications are involved in critical tasks that handle sensitive data and knowledge. Protecting the tasks and knowledge from unauthorized users is crucial to the safe operation of such systems.

Two types of security mechanisms may be imposed on distributed systems. They are: discretionary access control mechanisms to protect the system from unauthorized users depending on their user-ids or user group-ids [16]; and mandatory access control mechanisms that enable users cleared for different levels to access the system containing files at various sensitivity levels without violating security [17]. Enforcing the latter mechanism, even in operating systems, is still in the early research stages. As discretionary access control mechanisms have been extensively applied to operating systems, current approaches to security in distributed systems extend these techniques. Our attention will also be focused on discretionary access control mechanisms for distributed systems.

The discretionary access control mechanism employed in current distributed systems is to check if users have access permission to read from and/or write into files. If a user is at a remote site, the user's request is first transmitted to the target system and then checked for access permission. Alternatively, the access permission could be checked first, before transmitting the request to the target system. In a distributed environment, it is also possible to enforce discretionary access rules across the machines.

One of the problems with the current approaches to security in distributed systems is that they are not sufficient to solve the inference problem. The inference problem deals with the situation where users read from several files, each of which they are authorized to read, combine all of the information read, and infer information that they are not authorized to know. This inference problem is illustrated in the following scenario:

- File F1 has names of employees, file F2 has job titles, and file F3 has salaries. Each item in any of the files is identified by the employee’s social security number.
- The user has access to the employees’ names and job titles but not to their salaries.
- If a user knows the job titles, he can deduce the corresponding salary.
- The user reads files F1 and F2.
- Although the user cannot read file F3, he can now deduce the information in this file.

Complex mechanisms are needed to solve the inference problem in a distributed environment because the files could be distributed across several machines. We will now discuss the techniques that could be used to solve the inference problem.

**AI Applications in Security**

Two approaches are currently being taken to solve the inference problem in a nondistributed system [18][19]. They are illustrated below with respect to the scenario described earlier:

- Enforce a rule that does not allow a user access to file F2 if he does not have access to file F3. Alternatively, if a user has access to file F2, he should always have access to file F3. Whether the former or latter approach is taken depends on the discretionary security policy that is enforced. In either case, the discretionary access rules must be checked for consistency.
- No consistency checking is performed. Instead, before permission is granted to access file F2, it is checked first whether by granting the user access to F2, he may acquire unauthorized information. If so, the access is denied.

Currently, logic programming techniques are being investigated for both solutions. In the first solution, the consistency checker should have access to all of the rules. In the second solution, the system has to maintain history information on which files have been accessed and by whom.

In a distributed system, either one of the solutions discussed above can be extended. However, the problem here is that, as mentioned before, the files are distributed across different machines. The rules could also be distributed across the machines. Because the automatic planning component will have access to all security constraints, it will only generate distributed execution strategies that return results that the user is allowed to access. A user cannot submit a request to the distributed system that derives information that the user is prohibited from accessing directly. However, the user may submit several requests and use their responses to infer secure data. To prohibit this approach, a distributed system must “remember” earlier requests from the user to determine if the user will be able to infer unauthorized information from the current and previous requests. Such a distributed system is described in [20].

**Conclusion**

In this paper, we have discussed the design issues that relate to three of the functions of a front-end subsystem of a host in a distributed environment. They are:

- User interface
- Interconnection
- Security

Modern user interfaces ease the burden placed on users by providing facilities for browsing, command formulation, and automatic planning. For each of the above facilities, we have proposed possible approaches for its design. A special case of automatic planning is interconnecting heterogeneous systems. Here, we discussed the current trends and proposed two approaches using AI techniques.

Security is becoming increasingly important in distributed systems, as they are being used for military, process control, business, and financial applications. We discussed
the limitations of the current security control mechanisms and proposed possible solutions.

Traditional approaches to the design of user interfaces, interconnection of heterogeneous systems, and security have been beset by some problems. In this paper, we suggest that alternative approaches to solving some of these problems (using AI techniques) are needed if companies are to take advantage of the market opportunities that will become available in distributed systems.

References


Bhavani M. Thuraisingham received the B.Sc. degree in mathematics and physics from the University of Sri-Lanka, the M.Sc. degree in mathematical logic from the University of Bristol, United Kingdom, the M.S. degree in computer science from the University of Minnesota, and the Ph.D. degree in computability theory and recursive functions from the University of Wales, Swansea, United Kingdom.

She is a Principal Research Scientist at Corporate Systems Development Division, Honeywell Inc., and an Adjunct Professor of Computer Science, University of Minnesota. Her research interests are in intelligent database management systems, distributed processing, and database security. Previously, she worked as a Senior Analyst at Control Data Corporation, designing and developing computer networks. She has also served as a member of the faculty in the Department of Computer Science, New Mexico Institute of Mining and Technology, and in the Department of Mathematics, University of Minnesota.

Dr. Thuraisingham is a member of ACM, IEEE Computer Society, and Sigma Xi. She has published many technical papers in various journals and conferences in the areas of database security, AI applications, distributed processing, and computability theory.

James A. Larson received the B.S. degree in mathematics from Utah State University and the M.S. and Ph.D. degrees in computer science from Washington State University, Pullman, Washington.

He is a Senior Research Fellow at the Corporate Systems Development Division, Honeywell Inc., and an Adjunct Professor of Computer Science at the University of Minnesota. He is an authority on database management systems and user interfaces. His research interests include all aspects of database management systems and user interface management systems. Previously, Dr. Larson was involved in the design of integrated automated office systems for Olivetti, Italy, and research in database management systems for Sperry Univac. He was also a consultant to Systems and Informatics in Switzerland.

Dr. Larson is a member of the review committee for Benjamin-Cumming's series on database management systems, IEEE Computer Society, and ACM.

He is the author of numerous technical papers which have appeared in various journals and conferences, IEEE tutorials, and textbooks. His tutorial on distributed database management systems, given at various conferences on databases and distributed systems, is very popular among computer scientists and engineers.