Intelligent Resource Allocation in Distributed Collaboration

by

George S. Dolina

Submitted to the Department of Electrical Engineering and Computer Science

In Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

And Master of Engineering in Electrical Engineering and Computer Science

At the Massachusetts Institute of Technology

May 21, 1999

[June 1999]

Author .....

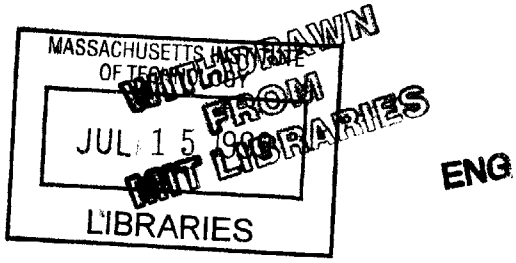
Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by .....

Howard E. Shrobe
Thesis Supervisor

Accepted by .....

Arthur C. Smith
Chairman, Department Committee on Graduate Theses



Intelligent Resource Allocation in Distributed Collaboration

by

George S. Dolina

Submitted to the Department of Electrical Engineering and Computer Science

In Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

And Master of Engineering in Electrical Engineering and Computer Science

At the Massachusetts Institute of Technology

May 21, 1999

ABSTRACT

This thesis examines the topic of collaboration, applied in a distributed computing environment. A design strategy is outlined, as well as an implementation of a Knowledge-Based system built in Joshua [1,2] is examined. The system chooses the "best" resources to use for rendering the desired collaborative services. The main criteria which determine what is "best" are utility and cost. The system considers the projected utility of a service and the cost associated with the resources needed to support it, to chose the best service.

Thesis Supervisor: Dr. Howard E. Shrobe
Title: Associate Director, MIT Artificial Intelligence Laboratory

ACKNOWLEDGEMENTS

This thesis describes research done at the MIT Artificial Intelligence Laboratory. Support for the MIT Artificial Intelligence Laboratory's artificial intelligence research is provided in part by the Defense Advanced Research Projects Agency of the Department of Defense, under contract number F30602-97-2-0013.

This thesis wouldn't be possible without the wisdom, guidance, and support of Howard Shrobe. I am grateful to Kalman Reti for helping me get my hardware working when I needed it the most. I thank my brother Peter Dolina Jr. for his insightful comments and input. I thank Elissa Arling for her support and understanding through some very busy times. Above all, thanks to my parents for their never-ending help and support.

1 Introduction

The objective of this thesis project was to design, implement, and evaluate a knowledge based system for choosing the optimal use of resources, which can be used to render concrete services based on abstract requirements. The goal of the system is to dynamically render concrete services in response to abstract requests. The idea is illustrated in Figure 1.

The system focuses on the services and resources used for collaborative problem solving in a distributed computing environment, a frequently seen domain in today's work environments. While such a system could be implemented in various ways, we will examine a possible implementation in Joshua [1, 2] and Common Lisp on a Symbolics Lisp Machine [3]. The following sections familiarize the reader with

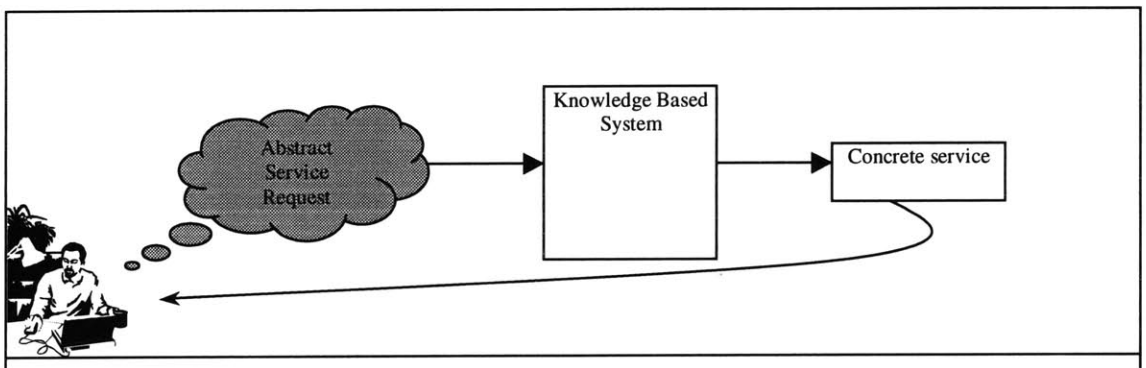


Figure 1

the background of the research, and then explore the design and implementation of the system.

1.1 Overview

Today's work environments are complex. Many resources are underused, or paradoxically overused. They are miss-managed, resulting in decreased productivity and increased costs. This problem is far reaching, and deeply encompassing. It manifests itself in most large groups of people sharing resources. Fred Brooks [4] has described many examples of things turning terribly wrong in large groups

of people working together towards a common solution. He has extensively examined the domain of software design. Most software systems built these days are too large to be built and maintained by individuals, or even small groups of individuals. It has been shown that increasing work force alone is not the magic solution to large problems [4]. In order to achieve current complex goals, increasing numbers of individuals must collaborate in a team effort. These teams must be coordinated to work well together. To achieve this sub-goal, the teams have many resources at their disposal, to help them collaborate. Collaboration is a daily task of many people, and yet often the available means are not utilized to their maximum potential. The utilization of the resources for collaboration itself presents a formidable task. To achieve maximum efficiency, resources need to be used "optimally." This thesis examines the utilization of these resources and the possible interaction and cooperation of the workers in a collaborative environment.

1.2 Collaboration

As stated above, collaboration is a key task of people working together toward a common goal. It involves complex interactions of independent people coming together for a shared purpose. Collaboration is based on a shared need to solve a common problem and is enabled by shared knowledge and understanding of the problem domain; the collaboration is driven by the current state of the problem solution; the attempt to solve the problem is constrained by the capabilities, goals, and workload of the current set of people available. There is no fixed process which will work every time and for all people. Collaboration can be carried out in many ways, such as meetings, discussions, videoconferences, phone conversations, email etc. Despite the many versions of collaboration, it can be categorized into abstract categories.

Collaboration can be carried out synchronously or asynchronously. At an abstract level, collaboration is really information sharing and cloning. The collaborators either have information which is useful for others, and share it, or need information which can be cloned from other collaborators. This breakdown is shown in Figure 2. The significance of this breakdown will be seen later when specific services which enable collaboration are examined in sections 2.4 and 2.6. The best choice of a collaboration method depends on many criteria.

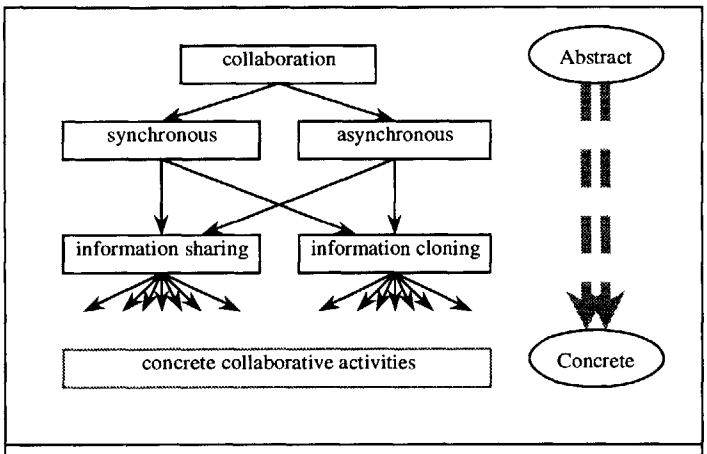


Figure 2

At this point we need to clearly see the difference between an *abstract* service and a *concrete* service. An abstract service is something the user wants to do (such as print out a document, get feedback about a project, find background information for a presentation) but he isn't really sure *exactly* how he will do it. A concrete service on the other hand, is a well defined set of actions (such as sending a postscript file to a specific printer; holding a teleconference with Joe Brown at 3pm on Monday the 24th from office 803 using the computer Polk; downloading file named *background.doc* from file server at IP address *18.251.3.92* using the user name *guest* and password *hello*) which use a specified set of resources.

1.3 Knowledge-based Collaboration Webs

The Knowledge-based Collaboration Webs (KBCW) project[5] seeks to facilitate new ways of setting up and supporting collaboration. The research described in this thesis falls within the larger KBCW project. The project involves integrating several separate lines of research within the MIT AI Lab in order to support a single unified vision. This vision involves computer mediation of broad area collaborative problem solving using natural language and other natural forms of interaction. In the long run, people will be interacting with one another by natural means with the computer "listening in" and "looking on;" the computer system is responsible for continuously constructing an ever-richer representation of the solution and of the problem solving process. The computer system plays the role of a facilitator that adopts different styles of interaction management depending on the problem-solving context. The system understands the advertised capabilities of each individual and other resources in the environment. The project described in this thesis fits inside the framework of dynamically enabled collaboration. The intended task of the system is to keep track of available resources and suggest the "optimal" way of using them to support collaboration by rendering concrete services in response to abstract requirements.

1.3.1 Related Research

The area of collaboration is widely researched because of its recognized benefit to most problem solving efforts. The following projects relate directly to the task of allocating resources for collaboration, and also to the greater KBCW project. The summaries presented bellow reflect the views of the participants in the projects.

1.3.1.1 *Jini*

The underlying goal of Jini™ [6] is to enable users to access a wide variety of services easily and transparently. Sun Microsystems designed the Jini™ system to provide simple mechanisms that enable devices to plug together to form an impromptu community—a community put together without any planning, installation, or human intervention. Each device provides services that other devices in the community may use. These devices provide their own interfaces, thus ensuring reliability and

compatibility. The available services are resources which can be used by the users in the impromptu network.

To use a service, a person or a program locates it using the lookup service. The service's interface is copied from the lookup service to the requesting device where it will be used. The lookup service acts as a switchboard to connect a client looking for a service with that service. Once the connection is made, the lookup service is not involved in any of the resulting interactions between that client and that service.

Jini is built on the Java language. It is the key to making Jini technology work. Devices in a Jini network are tied together using Java Remote Method Invocation (RMI™). By using the Java language, a Jini system is secure. The discovery and join protocols, as well as the lookup service, depend on the ability to move Java objects, including their code, between JVMs.

Jini technology not only defines a set of protocols for discovery, join, and lookup, but also a leasing and transaction mechanism to provide resilience in a dynamic networked environment. The underlying technology and services architecture is powerful enough to build a fully distributed system on a network of workstations. The connection infrastructure is also small enough to make it possible to create an impromptu network out of simple devices such as home entertainment devices.

1.3.1.2 *Villa*

Villa[7] is a system based on an asynchronous event based toolkit, designed to build scalable collaborative application over the Internet. It is implemented using the Java language to provide a set of mechanisms and functionalities which can be used to build new collaborative applications, and to enable existing applications for collaboration. Villa supports both synchronous and asynchronous collaboration. Since the requirements of Internet applications vary widely, Villa provides a configurable framework and infrastructure on which such applications can easily be built. The Villa research also studied the requirements of various collaborative applications over the Internet, and provided an extensible and lightweight set of mechanisms on which such applications can be built.

The Villa project aims to modify underlying resources (connectivity), to better enable collaboration over the Internet. It has focused on three communication channels: data, audio and video. These are used to establish four broad classes of collaborative applications: (1) conference applications, (2) business applications, (3) collaborative application development environments, and (4) generic applications.

1.3.1.3 *Reusable Agent Intelligence Software Environment*

IBM Research has considered practical software design requirements for rule-based intelligence in the next generation of commercial information agents. Besides basic inferencing, these requirements include: Embeddability, reusability, user-friendly authoring of rules, communicability of rules, flexibility in inferencing control strategy and performance, and extensibility of representation and reasoning. They developed an architecture that fulfills those requirements to a substantial degree: RAISE (Reusable Agent Intelligence Software Environment). RAISE provides building blocks for embeddable agent intelligence. It is founded upon a declarative representation and clean semantics, equipped with a simple yet powerful approach to procedural attachments. This results in highly pluggable components for inferencing, authoring, and communication. RAISE enables high reusability of both code and knowledge while embedding rule-based intelligence enhancements in three prototyped information agent applications: personal messaging, newsgroup filtering and handling of customer service support, and collaborative news service in Lotus Notes.[8]

1.4 Intelligent Resource Utilization in Distributed Collaboration

Taking a step back, and looking at the information before us, we can see clearly that enabling collaboration in a collective work environment is critical. The task of the system described in the sections below is to combine the knowledge of collaborative processes and resource availability, to help users utilize these resources optimally in order to achieve their goals and satisfy certain requirements. The system integrates knowledge of *resources*, *collaboration*, and the *way resources are used* for collaboration. These three key issues are examined in the following section.

2 Design

The main criteria used in choosing the best service to use are utility and cost. The system works by polling available resources, and obtaining necessary information from the user. The user inputs this information as abstract requests for service. With this information, the system will then reason about possible services that are supported by the available resources and satisfy the user's abstract service requirements. Very frequently, the available resources will support several services which could satisfy the user's request. How will the "best" one be chosen? The system will calculate the utility and cost for each possible service and return the one with the highest ratio of utility per cost. This optimization aims to balance the load on all resources evenly.

The rationale behind the design of the system is broken down into six sections. The *utility* section describes what we mean by utility and how it is used in the system. We next examine *cost* as a factor which is used in combination with utility to optimize the system. The section about resources examines the types of resources that are important to keep in mind, when reasoning about collaboration. The subsequent section describes the types of abstract and concrete services which can be enabled by the available resources. We conclude with a section about reasoning, which describes the reasoning that should be embodied in a system which supports collaborative services by utilizing the appropriate resources.

The design section describes a system which will satisfy the objectives of the system, as outlined in the introduction. These ideas explore the reasoning needed to build such a system. However, not all of these ideas are implemented. The implementation section examines in detail the implementation of the system.

2.1 Utility

Maximizing the utility of actions to individuals has been the pursuit of philosophers for ages [9]. Utility has been used as a metric for determining the usefulness of actions. It is a measure of how well a

service satisfies a given set of requirements. The main problem solving strategy of the system is to maximize utility. This is how the system distinguishes between competing available services.

The notion of utility is unique to each user. Everyone values different properties in different ways, but the system needs to quantify it on a universal scale. The system needs to determine what notion of utility the user has. To achieve this, we have a set of utility criteria which the user can choose for his request. The utility criteria vary for different abstract classes of services, each one appropriate to the concrete services which fall into the abstract service class. Each concrete service is also described by a set of utility criteria, which determine how well a given service can satisfy a specific request. Utility functions are used to calculate the utility of each possible service, given the utility criteria of the user and the utility characteristics of the service. The simplest utility function can have a linear scale for each possible parameter, such as speed and reliability. (For example, if a resource offers a reliability characteristic of A1 and a speed characteristic of A2, and the user needs reliability of level B1 and speed of B2, the utility of using the resource will be (A1*B1)+(A2*B2).) The utility criteria and functions vary, depending on the service being requested.

A more custom tailored approach is to determine the utility function for each user, and use that function during a session with the user. We are assuming the user is "smart" and knows what he wants. The system takes the responses at face value and works with them. This keeps the utility functions simple, but the system also lacks the ability to think "around" the user, and choose what is really the best service, if the user cannot choose his utility criteria well.

The utility characteristics of resources are dynamic properties. They change in response to the changing environment. A printer which becomes busy will provide a lower speed utility than when it is idle. Updating of utility characteristics can be viewed as overall knowledge maintenance in the system, just like adding and removing resources from the system.

2.2 Cost

Every resource has a cost associated with it. The cost of a service is the sum of the cost of resources which are needed to support it. We use it to compute the ratio of utility to cost for each service.

The cost of a resource is made up of inherent material cost and the desirability of the resources to others. The inherent cost is a fixed value, mostly dependent on the material value of the resource (such as purchase price). The desirability of the resource to other users determines the second part of resource cost. It is governed by supply-demand economics. As the demand for a resource increases, so does its cost. The goal of this scheme is to evenly distribute the usage of resources in the environment. The desirable resources are expensive, and thus less likely to be overused. The cost is a variable property like the utility. It is dynamically updated as conditions within the environment change.

2.2.1 SEREM (SErvice-REsource Money)

The idea of cost naturally leads to money. Users are allocated a certain amount of SEREM to pay for services, according to their rank and standing in the environment. The assignment of SEREM is analogous to budget assignments in work environments. Users get a budget of SEREM, use it as necessary, and then reapply for more.

This arrangement prevents overuse of the most desired resources. Due to supply-demand economics these will be the most expensive, and only a select group of users will be able to afford them. However, a group of users might pool their SEREM together to buy more expensive services. As an example, a large conference room is an expensive resource, because it can be used for many different activities (hence its desirability drives up its price). A single user will not be able to buy the room just for himself. A group of users however, will be able to collectively buy the services supported by the conference room.

Users receive allocations of SEREM through regular budget assignment, but also for providing services. When the users aren't using the system (requesting services) they are considered resources. As

such, they also have a cost. When the users are used to support a service, they are paid in SEREM, hence enlarging their own SEREM allocation.

2.3 Resources: description

The system needs to be aware of available resources in the environment. The richness of detail in any environment will guarantee that a system that delves too deeply into the details will be inefficient, slow and probably useless. The resources considered in this system are a broad concept, but must also be constrained to avoid getting buried under unnecessary detail. These resources are needed to enable services that support collaboration. It is important to keep a clear conceptual boundary between all resources, and those which are used for collaboration. It is unnecessary to keep track of everything, but at the same time some certain core knowledge about the available resources is vital. The resources we are interested in include hardware with its accompanying software, people, locations, and time. The people in the environment are also the intended users, thus the system knows about the resources that the user has at his disposal as well as other resources in general. The diagram in Figure 3 shows a representation of necessary and sufficient knowledge about resources, which will enable reasoning about it.

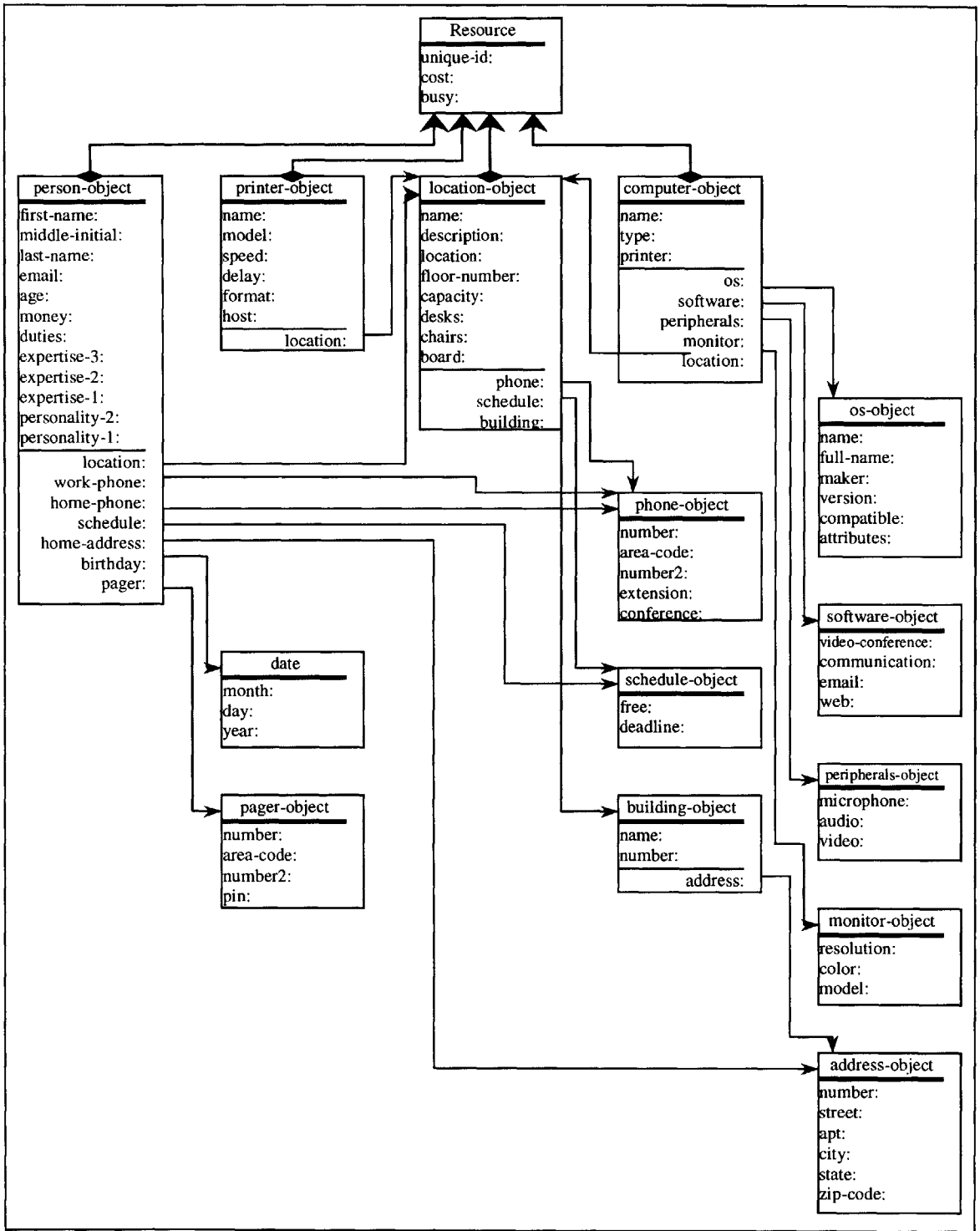

Resource
unique-id:
cost:
busy:
person-object
first-name:
middle-initial:
last-name:
email:
age:
money:
duties:
expertise-3:
expertise-2:
expertise-1:
personality-2:
personality-1:
location:
work-phone:
home-phone:
schedule:
home-address:
birthday:
pager:
printer-object
name:
model:
speed:
delay:
format:
host:
location:
location-object
name:
description:
location:
floor-number:
capacity:
desks:
chairs:
board:
phone:
schedule:
building:
computer-object
name:
type:
printer:
os:
software:
peripherals:
monitor:
location:
os-object
name:
full-name:
maker:
version:
compatible:
attributes:
phone-object
number:
area-code:
number2:
extension:
conference:
software-object
video-conference:
communication:
email:
web:
date
month:
day:
year:
schedule-object
free:
deadline:
peripherals-object
microphone:
audio:
video:
pager-object
number:
area-code:
number2:
pin:
building-object
name:
number:
address:
monitor-object
resolution:
color:
model:
address-object
number:
street:
apt:
city:
state:
zip-code:

Figure 3

2.3.1 Representation

Representation is the essence of programming[4,10]. It is therefore important to capture the important details in a representation which conveys them efficiently and naturally. Figure 3 shows that the system represents resources in the computing environment in a semantic-net structure, made up of a hierarchy of objects. The objects portray a relatively high level view of the computing environment, hiding the details of network connectivity, medium, protocol, and bandwidth. These could however be incorporated by extending the representation further. We are using the higher level approach in the interest of simplicity, and the availability of the top level knowledge. This information could be gathered from a network administration authority.

The semantic net is formed by implicit reasoning assumed in the organization of the resources. A person is linked to a location object, which specifies the person's location, but also all of the equipment at that location, and thus the person is assumed to have access to all of that equipment. This coarsely assumes that if a person has access to a location, he also has access to all of the equipment there.

The top-level class of resource has four subclasses: person, printer, location, and computer. These are further composed of other resource objects. The reason behind the choice lies in how collaboration is carried out. This particular representation makes it easy and natural to think in terms of collaboration.

The hierarchy of resources is only two levels deep, with a top-level object called "resource" and second level resources that inherit from the "resource" object. The second level objects have various slots that can hold other objects as values. In the diagram these slots follow the second divider line in the object frame. This is a natural way of representing how resources are related in the domain which is being modeled. Care was taken to prevent loops from forming in the representation net. The main reason for this design is the task at hand: allocating resources to render services for collaboration. This particular representation is well adapted for the task. The listing of the Joshua code of the representation can be found in the Appendix.

2.4 *Services for Collaboration*

Any computing environment offers a myriad of possible resources and services. Given the large number of available services, it can be difficult to choose the best ones for the task. These concrete services need to be classified into more abstract categories, to make it easier to reason about them in the context of collaboration. We need the ability to quickly reason and recommend the best service or combination of services to the user.

Because the goal of the system is to satisfy abstract service requests with concrete services, using available resources, we need an abstract view of services used for collaboration. We have chosen to break down the set of concrete services into the abstract categories shown in Figure 4. It should be apparent that this diagram is similar to Figure 2, which describes collaboration. We're drawing a parallel between the

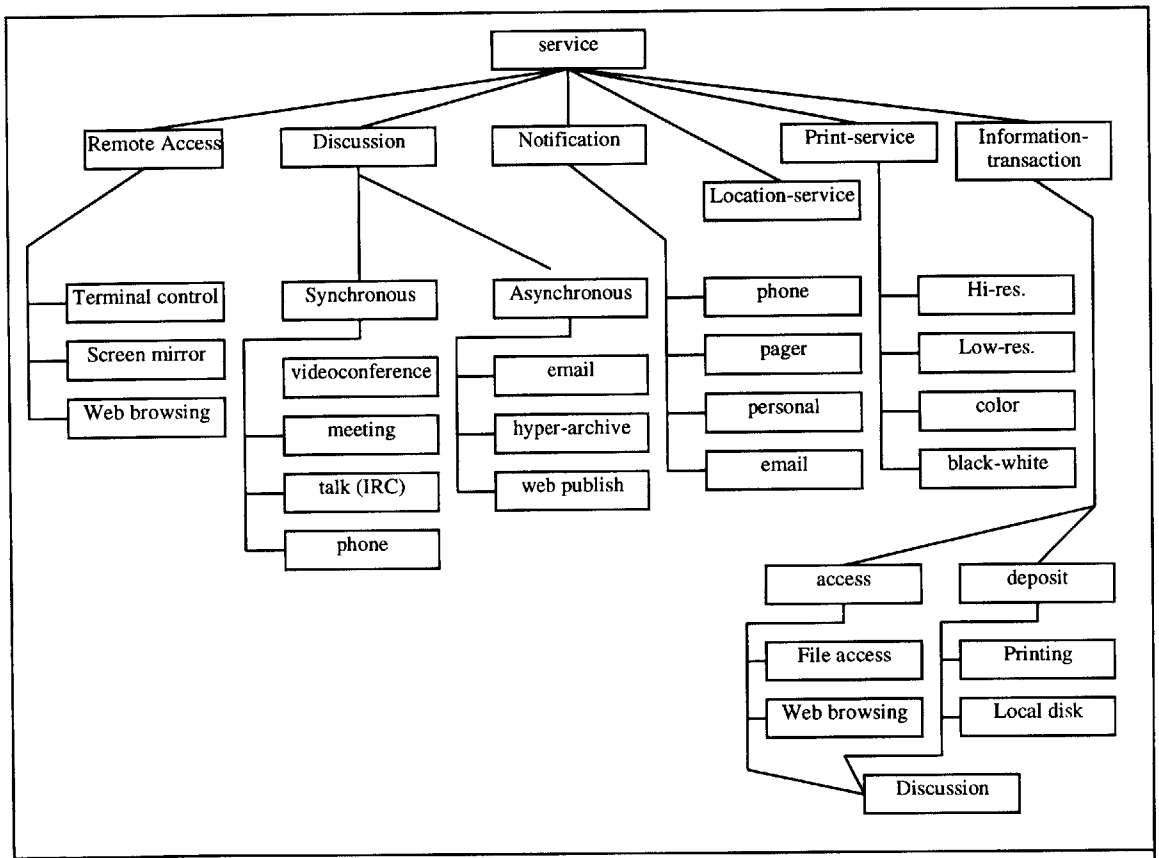


Figure 4

types of collaboration, and the concrete services which are used to support them.

The six, top-level classes include: remote access, discussion, notification, information transaction, location service and print service. These abstract services can be used as a basis for most collaboration in a distributed computing environment. It is often possible to have the same concrete service satisfy different abstract requirements. This means that the same concrete services can be chosen as a response to different abstract requests.

2.4.1 Discussion

Discussion is the exchange of information between two or more parties. A participant either brings new information to the discussion, which can then be used by the other participants and taken away, or the participant simply attends the discussion to gather new information. For any discussion, there must be a set of participants, locations, and resources. A location could be an office, a meeting room, or simply a server to which all participants remotely connect.

Discussions can be either synchronous or asynchronous. A synchronous discussion imposes an additional restraint, time. The discussion must take place at a certain time, for some duration, and the interaction between participants has a limit on latency. As latency grows, the discussion becomes asynchronous.

The concrete services for synchronous discussion include video-conferencing, telephone conferencing, meeting, and using online chat (ICQ, AOL Instant Messenger etc.). An asynchronous discussion can be instantiated by mail, email, faxing, web publishing and other services.

2.4.2 Information Transaction

This large abstract category could be viewed as the super class of the other service categories. After all, collaboration really involves the exchange of information, whether it is data files or programming commands. The top-level categorization would be of little use in the current context. Instead, information transaction will be a more literal interpretation of the term. It is made up of information access, where a

user is trying to get some information, and information deposit, where a user is delivering information for use by others.

Information access can occur through file access, browsing the web, and just collecting data in various ways. Information can be deposited in many ways. These include storing data in a server, publishing it on the web, saving it to a local disk (on a local computer), and printing out a hardcopy.

2.4.3 Location Service

The location service is used by users to locate resources in the environment. While the system should be able to suggest to users what resources to use, the location service leaves an option for the user to reason on his own, and only use the system as a means of keeping track of the resources.

The location service is truly abstract, as it relies on many other services to locate resources throughout the system. A few concrete services used by the location service include finger, gopher and querying the namespace object [11] among others.

2.4.4 Notification

The notification service is intended as a one way, relatively fast information delivery, which will get the receiver's attention. It is in fact a form of information transaction, which was described above in section 2.4.2, but because of its frequency of use, it receives special attention when it is used to notify.

Different forms of notification serve different purposes. The main descriptive characteristics of notification are latency and content capacity. Concrete services considered for notification include phone calls, personal notification, paging (using a pager), email and cellular phone calls.

2.4.5 Print Service

Print service has been mentioned above as part of information deposit. It was given its own abstract category because of its importance and frequency of use. Implicitly a user thinks about printing a

document, rather than depositing it as data. Sometimes a user might want to deposit some information, and printing it out could be the best method, given the user's requirements.

2.4.6 Remote Access

Remote access encompasses all services which deal with a user accessing a computer physically removed from his location. These services can be useful when a user is collaborating with another person at the remote location, and he needs to get direct access to the task. This includes dialing up to a server, using telnet, browsing the web, controlling a computer remotely, and mirroring the screen of a remote computer.

2.5 Request for Service

The system recommends to users the best way to use available resources to render services based on abstract service requests. These requests come directly from the user and must contain information which is used to decide which services should be rendered and what resources they should use. The content of a request depends on what main class of service is being requested. The actual format of the requests will be examined later as a part of the system implementation.

2.6 Reasoning: collaborative services

The system needs to have a knowledge of the resources available in the system, but it must also know how these resources support services. This is the key knowledge in the system, which will enable it to consider what is available and what is required, and use that information to calculate the utility and cost of possible services. The following sections describe what the system should consider when reasoning about the different classes of services. The prevailing theme in this section is the tradeoff between utility and cost. The cost is relatively easy to calculate, as it is the sum of the costs of the resources needed to support a specific service. The utility considerations are much more subjective, and are discussed as such.

2.6.1 Asynchronous Discussion

An asynchronous discussion is used for the exchange of information in a two-way fashion. The participants do not have to be available simultaneously. The discussion's purpose is to get feedback or information about a new idea, suggestion, or thought. It can really be classified as information transaction, when viewed as a more abstract idea. However, it is a specialized form of information transaction, and will be treated separately. The utility criteria for an asynchronous discussion include: the topic of discussion, the speed and latency, the content of the discussion, and privacy requirement.

The utility criteria necessarily overlap. The topic of the discussion determines which people are good candidates for the discussion. The people "resources" have a slot describing their expertise, so that a person with an expertise in the field which is the current topic can bring more utility to the discussion. The speed considerations also depend on a person's characteristics. Some people respond to email much faster than others, and workers who are busy will respond slower than when they are free. These properties of people are stored in the resource knowledge base.

The content of the discussion not only affects the choice of participants, but also the means of communication. If graphics are prevalent, for example, simple text email will provide little utility. Instead the user will have to consider services such as faxing, web publishing, or using email with attachments.

Another factor which should be considered, but is inherently difficult to quantify, is the personality of a potential participant in a discussion. To really fine-tune a discussion, it would be useful to know how willing to communicate the people are; how friendly and accepting they are to disturbances; how well they can communicate their expertise. These factors are all very subjective and difficult to quantify.

A request for asynchronous discussion will generally be satisfied by one or more of the following abstract services: locating a user, notification, email, fax, hyper-archive, newsgroup, voice mail, web browsing and web publishing.

2.6.2 Synchronous Discussion

A synchronous discussion is necessary when the user wants to get feedback in real-time, or when he has an idea that cannot be expressed in writing or is still too vague to express coherently in writing. Often a very private and sensitive topic will also require a synchronous discussion. What is the reasoning involved? First the system needs to establish the fact that synchronous discussion is in fact necessary. This is normally accepted if the user asks for the synchronous discussion directly. If it is necessary, what other factors need to be examined? The topic of the discussion is important, as well as time considerations.

The time considerations for a synchronous discussion are more stringent than for the asynchronous. The system needs to know when and for how long to hold the discussion. The utility criteria for the time allow the user to specify a time but also to assert how sure he is of that particular choice. Often a person will schedule an event, without really needing it at the chosen time. The flexibility factor will allow the user to specify if he is certain about a time.

The choice of participants is also important, and probably more so than in the asynchronous case. The personal traits of people are more apparent in a synchronous discussion, which usually involves more person-to-person interaction.

The utility calculation for synchronous discussion involves the time requested, the topic, the speed and time request, and the potential participants' personal properties and time schedules.

The user requesting a discussion might be a supervisor, who wants to establish collaboration between other users. In this case, the supervisor establishing the discussion is the user, and the other people are simply the resources used to render the discussion. Since the system holds information about all known users, it will have enough information to find a solution. The utility will however be calculated relative to what the user wants, ignoring the possible preferences of the people involved. This is consistent with the idea of maximizing utility for the user. The system is only concerned about providing the best utility for the user. An important question does arise however. Will the system really render the "best" means of collaboration when viewed in the context of overall utility? The system should take into consideration the

utility preferences of the individual participants, and calculate the overall utility of the service as the sum of those utilities and that of the user (who initiated the service).

A synchronous discussion can be supported by the following services: meetings, remote screen (which is a subset of remote-access), remote screen in combination with voice (telephone, or dedicated software), voice alone (phone, internet phone), and teleconferencing (supported by various software/hardware configurations).

2.6.3 Information Transaction

When a user requests this abstract class of service, he will either ask to access some information, or ask to deposit it. In either case the system will need to know what kind of information it is (data file, graphics, hardcopy, text, sound, video etc.) and what information storage services are available. Services which are used for information transactions include: file server access, web browsing, discussions, and printing.

2.6.3.1 *Information Access*

To access information, the user needs to choose a topic, and a preferable medium for the information. From these requirements and information about available resources (data, documents, files, people etc.) and their properties (graphics file, sound segment, talkative person etc.) the system calculates utility and cost.

2.6.3.2 *Information Deposit*

When depositing information, the user must specify what kind of information it is, and what topic area it covers. This allows the system to update its knowledge, and later use that information when someone else needs it. The utility considerations for utility deposit include the topic of the information, the format, the medium it is currently in, speed requirements and the necessary persistence of the information. Some services which are used for information deposit include: discussions, file server storage, and printing.

2.6.4 Location Service

To locate resources throughout the environment, the system searches through its information about resources, and tries to deduce from it where the resources might be. In this process, utility and cost are not considered, as the system only provides information about resources, rather than actually use them.

Most frequently a user will try to locate people, who are the most dynamically moving resources in the environment. While other resources can also move around (such as portable computers, and relocated printers), their location information doesn't change as frequently as that of people.

To determine where a person is located, the system needs to know the current time, and some information about the person's movement habits, personality traits, and schedule. This information can be sufficient to determine where the person is located. For example, if the person's schedule indicates a meeting at the time the location request is placed, the system will conclude the person is located at the location of the meeting.

Although the location service has many useful applications, it also borders on violating certain privacy rights. The person might not want the system to know where he is at all times. The more information the system has about a person, the better able it is to locate him. In an extreme case, the user could have some tracking device attached, which would always inform the system of the person's location.

2.6.5 Notification

Notification is the delivery of information, rather than retrieval. The utility criteria for notification are urgency, content of the message, and the target person(s) to be notified.

The urgency determines how quickly and reliably the information needs to be delivered. High urgency services are also very distracting, so their associated cost is high, to discourage their frequent use for trivial matters.

The content of the notification is used to further calculate the utility of specific service options. If the content is large, for example, a pager will not be able to support it. On a deeper level, the content can

be used to determine who needs to be notified. The user might notice a situation which requires attention, but the user might not know who needs to be notified.

Whom should the user notify in an emergency situation? Staying focused on the system at hand, we will consider only issues which might arise in a computing environment. Possible mishaps should be captured in each user's "duty" property. Using it the system can reason about the content of the notification and the most appropriate person(s) to notify.

All three criteria (urgency, content, person to notify) really work together to choose the best service to support the notification. The requested person has a set of available resources which can render various notification services. The system can only choose from these when deciding on the final solution. Some of the specific services which support notification include: email, telephone, fax, voice-mail, regular mail, zephyr (Athena), cellular phone, pager, and personal notification. The final service recommendation will again be based on the best utility to cost ratio. The utility of a notification is based on who should be notified, how quickly they actually can be notified, and what the notification contains.

2.6.6 Print Service

To calculate the utility of print service, the system needs to know the user's requirement of printer location, speed, resolution, and color capability. The system knows about all printers available in the environment and also which ones the user can access. In calculating the utility of printing to a certain printer, the system checks how well the printer's advertised characteristics satisfy the user's request.

The computer used by a user is usually not "aware" of all the available printers. A user can print only to a printer which is configured on his computer. This adds another requirement into the reasoning about printing. We assume that setting up new printers takes too long, and can't be suggested as a service. The system can, however, check if the user has access to other computers, and which printers those computers recognize. This increases the pool of potential printers. The system can then tell the user to first move the data to the appropriate computer, and print it from there to the printer which has been chosen as optimal.

2.6.7 Remote Access

This service can be an intermediate step for a number of other services. For "pure" remote access, the user will need to do some work at a remote location. He will need an actual interaction with a computer, to execute some task at a location which is physically inaccessible to him.

The user's location will play a large role in the utility calculation. The location determines what resources the user has at his disposal (usually computers) that allow remote access. Additionally, the user needs to specify what he want he wants to accomplish (updating program, checking email, downloading data, etc.) through the access. These factors will be compared with the available resources' characteristics to determine the utility. Because remote access applications are not yet a widely standardized class of software, the final answer from the system will depend on what resources are available in the environment.

3 Implementation

We have explored the design of a system which can render concrete services in response to abstract service requests. In this section, we will examine one possible implementation of such a system, which does not support all of the reasoning outlined in the design section. It does, however, demonstrate the feasibility of such a system.

The system is implemented in Joshua, a reasoning system built in Symbolics Common Lisp. It is a knowledge-based system (sometimes called "Expert Systems"), and the main implementation strategy is

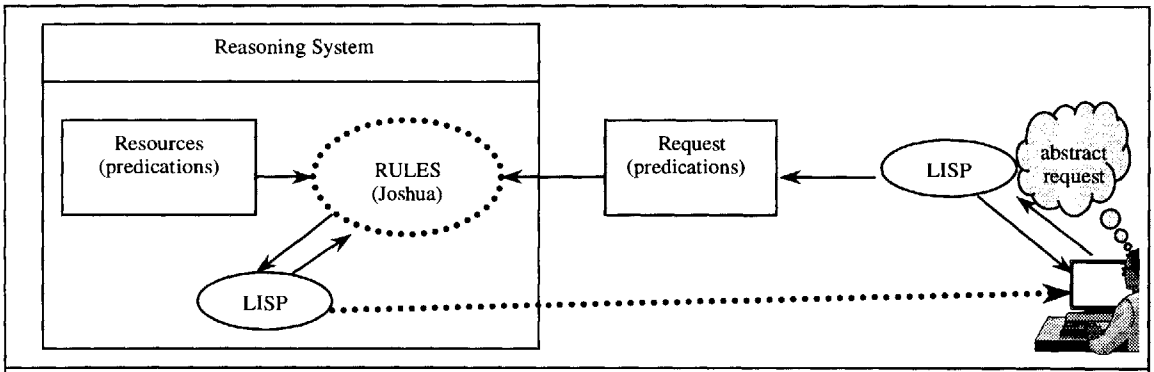


Figure 5

to tell the system "what to know" rather than "what to do."[12] The system uses backward chaining rules to search through an assertional infrastructure describing the computing environment, and calls various utility functions to calculate the utility of possible services. The final result is a recommendation to the user, telling him what resources to use to render the services which best satisfy his abstract request. This idea is illustrated in Figure 5.

3.1 What does the system know?

The system has information about a sample set of resources available in the environment. Its primary knowledge however is how resources are used to render concrete services out of abstract requests. This knowledge implicitly includes a taxonomy of services and service classes. The tree in Figure 4, on page 17, shows the conceptual organization of services. The system can be thought of as traversing this list, narrowing down on a specific service at the bottom of the tree. This tree is not always constructed as pictured in Figure 4. All of the services depend on certain resources. If the resources are not available, some branches of the tree can't be followed. Thus the tree is different for each computing environment, and each set of user requirements.

The system has to know about available resources in the environment. An object model lends itself well to representing resources. Each resource is an object with slots which might contain other objects. The diagram of this representation is shown in Figure 3, on page 15. The actual implementation of the system bypasses this representation and stores the information about the world explicitly in predicates, because the number of resources in the sample environment is small. The resource-object representation would be used with a larger sample world, and some amount of computation would be used to transfer the knowledge into the recognized predicates, or rules could be rewritten to directly access the information in the objects.

3.2 How does it work?

The reasoning is expressed in mostly backward chaining rules. It is described below, and a few rules are listed as examples.

The general approach to a problem is as follows:

1. Question the user.
2. Poll the available resources which can be used to render services which fall into the user's top level service request.
3. Calculate utility and cost of all services which can be rendered.
4. Return the "best" one(s). We assume the best service has the highest utility to cost ratio.

But this can be seen as really doing only utility and cost comparisons. The initial step of polling the available resources which fall into the user's service request category is really only a step to reduce the number of utility computations that will be needed later. It is clear that if all imaginable services were examined for cost and utility, most of them would have a utility of zero. For example, if a user asks for a discussion, printing a document will have a very low utility for him. Hence the initial step of narrowing down the search to only the services which fall into the top level category the user has requested, is a utility calculation eliminating all the services with a utility of zero.

3.2.1 Joshua

Joshua is an extensible software product for building and delivering expert system applications. It is a very compact system, implemented in the Symbolics Genera environment [1,2,12]. Its syntax is uniform, statement-oriented, and Lisp-like.

The core of Joshua is a rule-based inference language. Joshua has five major components: predications, database, rules, protocol of inference, and truth maintenance system. The knowledge based application discussed in this thesis is built around the predications, rules, and database.

Predications are ways of expressing knowledge. They are sometimes called *statements, facts,* or *assertions.* Joshua has a built-in virtual database, implemented as a *discrimination net.* Rules are ways of expressing and remembering relationships among predications, as well as procedural knowledge. For more information on Joshua consult [1, 2].

3.2.2 User Interface

The user interacts with a lisp function, which serves as a user interface. (Please refer to the Appendix for a code listing of the *user-input* function.) On a text display, it queries the user for an abstract service request. It begins by letting the user identify himself and choose one of the abstract classes of

services. Depending on this choice, the program next presents utility criteria which pertain to that class of service. A diagram representation of this scheme is shown in Figure 6.

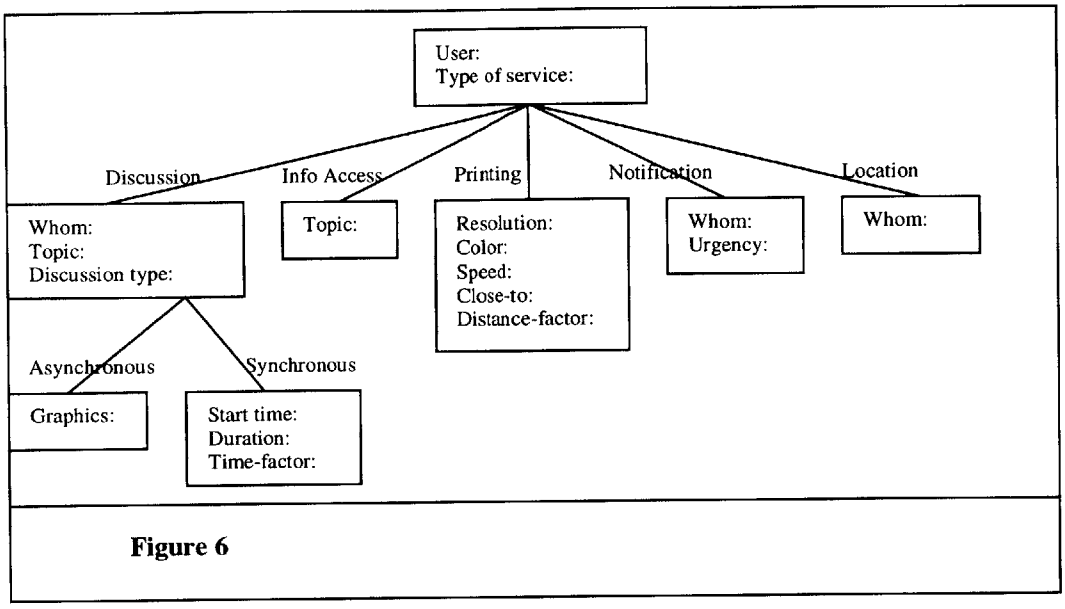


Figure 6

3.2.2.1 ***Discussion***

If the user chooses a *discussion*, the system further asks him with *whom* to hold it, what the *topic* will be, and what *type* of a discussion it should be (synchronous or asynchronous).

If the user chooses *asynchronous*, the system asks for a *graphics* requirement (whether the presentation of graphics will be important in the discussion). The possible choices are numbers between 0 and 4, with 4 corresponding to graphics being extremely important, and 0 to not important at all.

If the user chooses a *synchronous* discussion, the system asks for a time and a duration for the discussion, as well as a time-factor, which lets the user specify how certain he is of the selected time. To enter a time, the user can use most recognized time formats (ex: 6/12/99 12 p.m.). The Lisp function can recognize a variety of input formats, and converts them to a *universal-time*. The time-factor ranges from 0 to 6, with 6 offering no flexibility, and 0 having a flexibility of one month. The flexibility is only used to choose the start time of a discussion. The duration is fixed to what the user requested.

3.2.2.2 *Information Access*

When the user selects information access, as the top level service, the system asks for a topic of interest, and a speed factor. The speed factor indicates how important speed is to the user. We assume that if speed is important to the user, it means that *fast* information access is desired, as opposed to a high speed factor indicating that the user very strongly desires *slow* speed.

3.2.2.3 *Print Service*

When the user selects print service, the system further asks the user for his requirement of resolution, color, speed, and location of the printer. The quantity the user chooses for speed, is also used as the speed-factor, which expresses how important it is for the user that the potential printer does in fact provide the speed he requested.

The location is specified as "close-to" a known user. The default for this is the current user, but he can choose to send a print job to another user. This feature is useful, because the user doesn't need to know the locations of the other users to be able to send a print job to a printer close to them. Currently the system uses a person's office as his default location.

3.2.2.4 *Notification Service*

When the user selects notification, the system again asks for the person to notify (*whom*) and the urgency of the notification. The user can select an urgency between 6 (high) and 0 (low).

3.2.2.5 *Location Service*

When the user selects the location service, the system only asks for the person to locate. It does not consider any utility preferences, because the user is simply trying to find a specific resource.

3.2.3 Reasoning

Once the system has all the necessary input from the user, it converts it into Joshua predications and adds them to the Joshua database. The user-input function then *asks* the appropriate question of the

reasoning system (e.g. `(ask [answer-asynch-discussion ?user ?person ?service ?resource] #'say-query)`). A sequence of backward chaining rules is called depending on what information is present in the system, and finally a solution is returned. As the rules are called, they can call Lisp functions to execute some of the more complicated calculations, such as utility and cost.

3.2.4 Rules

The system "reasons," by following rules which conclude useful information. The diagram in Figure 7 shows how the system follows a set of rules to accomplish the top level goal of *answer-synch-discussion*, which is the final answer when the user requests a synchronous-discussion. The boxes

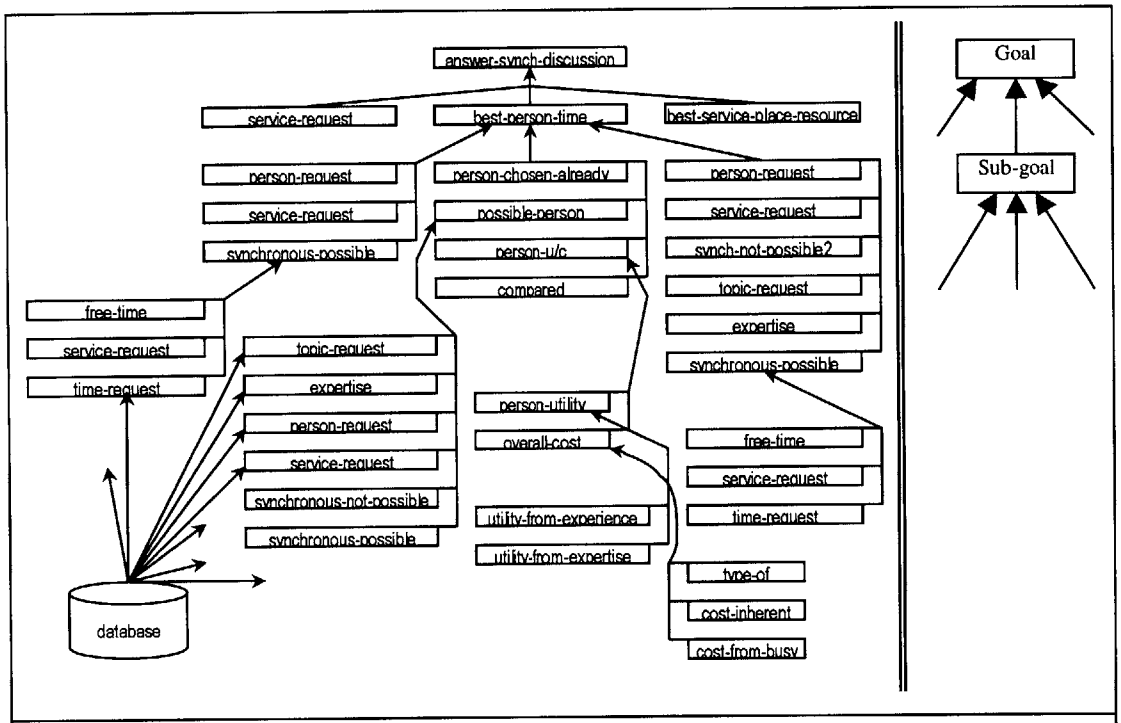


Figure 7

represent *goals* and the arrows represent rules which are used to reach the goals. The lines connecting boxes, but with no arrows, show which goals need to be reached in order to follow the rule upward, to reach the next higher goal. For any goal to be reached, all of its sub-goals must be reached first. This can occur it two ways: the predication of the sub-goal is known in the database, or additional rules can be followed lower down, to satisfy the sub-goals of the sub-goal in question. (This is standard backward

chaining). Figure 7 is only a partial graph of the rules, demonstrating how sub-goals lead to higher level goals, and finally to the desired answer. For a full listing of the rules please refer to the Appendix.

3.2.5 Utility Functions

The Joshua reasoning system uses backward chaining rules to perform most of the work (finding available resources), but has to call various Lisp functions to perform utility calculations. These calculations are simpler than might be imagined from the Design section. An example is shown below.

3.2.5.1 *Example: utility-from-experience*

The function shown below is used to calculate the utility of a person from his work experience. This function is used when reasoning about requests for any kind of discussion. The overall utility of a person is made up of the utility from expertise and the utility from experience.

```
(defun utility-from-experience-f (years)
  (cond ((<= years 10) (/ (^ years 2) 2))
        ((and (< 10 years)
              (<= years 13))
         (+ (utility-from-experience-f 10)
            (+ (* 10 (- years 10)) (* -1.5 (^ (- years 10) 2)))))
        (t (+ (utility-from-experience-f 13)
              (+ (- years 13) (* -.01569 (^ (- years 13) 2)))))))
```

What does this function do? We are claiming that the utility of a person can be equated to the area under the curve shown in Figure 8. The Lisp function shown above, calculates the area under the curve. We can see that the utility grows fastest for every year in the first ten years. In the subsequent three years it slows down quickly. After 13 years, the utility increases only very slowly each year. Other utility functions can be found in the Appendix.

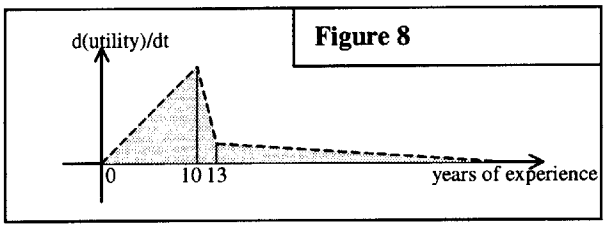


Figure 8

3.3 Example

The user selects options from a list presented to him. These include his identity, and a service class request. The system further questions the user depending on the service class selected. If the user is *gsdolina* and requests a *discussion* with *Hes* about the topic *lisp* of discussion type *synchronous*, the system further asks the user about the time, duration, and a time-factor, or how sure the user is of the time he chose. Given this input, and the availability of appropriate resources (*Hes*'s schedule shows free time), then the system will answer: "You can have a MEETING with HES in NE43-839 between 5/11/99 10:00:00 and 5/11/99 12:00:00 using NIL." However, if the requested person weren't free at the desired time, the system would check the schedules of other people who have an expertise which matches the user's request for topic. Then the system would calculate the utility/cost ratio for all of these people, and return the one with the highest ratio as the best person with whom to have a discussion.

3.3.1 Details

Let us trace through the above example in detail. The output of the system depends on the availability of certain resources. This example will examine the whole tree of possibilities stemming from a synchronous discussion.

1. The user is presented with the following display:

   Who are you: ***Gsdolina*** *Hes Rladdaga Blumberg Cvince Gregs Jcma Rhu Reti*

   Choose a service: ***Discussion*** *Info-access Location-service Print-service Notification Give-info*

The user has to click on a name and a service. The selected choice becomes bold. When the user chooses "discussion," further options appear:

With whom: ***Hes*** *Rladdaga Blumberg Cvince Gregs Jcma Rhu Reti*

Topic: Brainstorming Hacking ***Lisp*** *Management Money Politics Thesis Other*

Discussion Type: ***synchronous*** *asynchronous*

The user has to choose with whom he wants a discussion. The choices do not include the current user, as it would be rather boring (or troubling) if he had a discussion with himself. The topic of the discussion is also an option the user needs to choose. He can choose one of the listed options, or choose "other," and a new line will be presented soliciting the user to write in the topic. Finally, the user needs to choose whether he wants a synchronous (real time) or asynchronous discussion. When the user selects "synchronous," the system asks for more input:

When: enter a universal time

For how long: enter a time interval

*Time factor: 0 1 2 3 4 **5** 6*

Press <abort> to abort <end> to use these values

Here the user needs to pick a time and duration for the discussion. The time factor is a way of saying how sure the user is of his choice. A factor of 6 corresponds to an exact time. Hence the requested person must have a free time slot which exactly corresponds to the user's requested time slot. A factor of 5 allows 30 minute leeway in both directions of the start time of the discussion. The duration is always as requested. The other factors correspond to: 4—one hour, 3—four hours, 2—one day, 1—one week, 0—one month. When the user is satisfied with his choice, he will press *end*. Now the system has all the information that it needs from the user, which pertains to the *discussion* class of service. What happens next?

You can have a MEETING with HES in NE43-839 between 05/11/99 10:00:00 and 05/11/99 12:00:00 using NIL.

4 Conclusions

The study of this system has demonstrated a feasible system for helping users utilize resources in a more optimal fashion. It was built in the spirit of a vision, of a computer system observing and helping users work more productively, through more effective collaboration. A future system could be much more powerful, with more knowledge. In this system, knowledge directly translates to power. The more the system knows, the more it can do. If the system knew "everything it needs to know," it would be an immense asset in any work environment. This means that if the limit of knowledge grew to infinity (know where *all* the resources are, and what they are doing *all* the time), the system could be truly optimal, if its embodied reasoning was sound and correct.

It is not known whether such reasoning can be developed. However, it is possible to enable the system to have complete knowledge of the environment. Even with imperfect reasoning, such a system would be more powerful than any human user. It could hold much more accurate and updated information about available resources, which simply isn't possible for humans.

4.1 Evaluation

The system implementation meets the objectives set in the introduction. It recommends the optimal use of resources to support concrete services, based on abstract service requests. The system demonstrates the main ideas explored in the design section. Its practicality depends on the amount, and quality of the available knowledge. The current system has only limited knowledge of a small (9 people and some of their accessible resources) environment. Due to the limited amount of the knowledge, the current system is of limited practical use. If we view the system as only the rules and Lisp functions, then we can make the system much more useful, by simply supplying more data about the environment.

4.1.1 Scalability and Portability

The current system is not scalable to large work environments due to the design of the rules, and the Joshua database. The rules are not optimized for speed, because as mentioned earlier, the system is told "what to know" rather than "what to do." While this enhances the clarity and readability of the rules, it decreases the computational efficiency.

The Joshua database, which holds all the knowledge about resources, is sufficient for this implementation, but a faster database would be preferable for large scale implementations. A suitable interface between Joshua and such a database would have to be built.

4.1.2 Robustness

The current system is not sufficiently robust. It assumes all the data passed to it is correct. Its performance degrades significantly when incorrect data is given as input (e.g. the system could recommend to a user to print to a printer which is no longer available, if the environment data isn't updated properly). It is impossible to always provide an answer in response to a request. Since the amount of resources in any environment is finite, there must exist certain requests that cannot be satisfied.

4.2 Future Work

The current system could be improved and extended in numerous ways: The main component of the system, the reasoning (rules), can be optimized and updated; more detailed information about available resources will increase the scope of the problems the system can handle; automatic data acquisition will make adding and updating information about resources much faster.

4.2.1 Adding rules

Adding rules increases the system's capabilities to handle new and more complex situations. In particular, situations which involve combining current services would be very useful, and utilize a lot of the current system. For example, to *give information* to a person, the user could first *print* or *fax* some data to a printer or a fax machine, and then *notify* the intended recipient. The current system could be used for that,

but the user has to request both services independently. The reasoning used to connect *printing* (or *faxing*) and *notification* has to be done by the user. It would be possible, and useful to introduce rules which connect the individual abstract services into more complex and elaborate services.

Although we have a Joshua object representation of the environment, the system uses simple predicates to hold information about the environment (resources), due to the small size of the data. To enable the system to use the Joshua object representation, additional rules need to be written. They will translate the information about the objects into predicates which work directly with the current rules. The other option is to rewrite the current rules to access and reason about the objects directly.

4.2.1.1 *Learning*

At the conclusion of every session, the system could query the user about his satisfaction with the system's performance. The current rules have an *importance* factor that determines which rule is used, when multiple rules could be used in a given situation. For all positive feedback, the system could conclude that the rules it had used work well, and raise their importance factors. For negative feedback, the system would conclude that the rules that had been used aren't as good, and would lower their importance factors. In this fashion, the system would "learn" how to function better.

4.2.2 Data Acquisition

The current system has a small amount of knowledge about the environment. All of the data was gathered manually. This process is long and tedious. Automatic data acquisition would represent a great improvement in the system. It would enable the system to "know" about many more resources, and lead directly to dynamic updating of the environment data.

The system should have frequent updates of the environment data. Currently all the information is static, so a manual update is needed every time the environment changes (e.g. a new printer is added, a person moves to a new office etc.). Dynamic data updating would improve the performance of the system.

4.3 References

[1] Documentation Group of Symbolics, Inc., User's Guide to Basic Joshua. Symbolics Inc., Mar. 1990.

[2] Documentation Group of Symbolics, Inc., Joshua Reference Manual. Symbolics Inc., Feb. 1990.

[3] Documentation Group of Symbolics, Inc., Genera User's Guide. Symbolics Inc., Feb. 1990.

[4] F. P. Brooks, Jr., The Mythical Man-Month, Reading, MA., Addison-Wesley, Oct. 1995.

[5] P.H. Winston, R. Brooks, R. Davis, B. Katz, T. Knight, "Knowledge-based Collaboration Webs," MIT Artificial Intelligence Laboratory, Cambridge, MA. Proposal for BAA 97-09, Sep. 1997.

[6] Sun Microsystems, "Jini™ Technology Executive Overview," Revision 1.0, Jan. 1999.

[7] B. Mukherjee, S. Bhola, S. Doddapaneni, "Villa: An Event Based Middleware for Real-time Collaboration on the Internet," T.J. Watson Research Centeter, I.B.M. Yorktown Heights, NY., RC 20948 (92734), Nov. 1997.

[8] Grosof B. N., Levine D. W., Chan H. Y., Parris C. J., Auerbach J. S., "Reusable Architecture for Embedding Rule-Based Intelligence in Information Agents," IBM Research Division RC 20305 (89713), Dec. 1995

[9] J. S. Mill, "Utilitarianism," in Classics of Moral and Political Theory, M. L. Morgan, Ed. Indianapolis, IN., Hacket Publishing Company, 1996, pp. 1101-1139.

[10] R. Davis, H. Shrobe, P. Szolovits, "What is a Knowledge Representation?," AI Magazine, Spring 1993.

[11] Documentation Group of Symbolics, Inc., Site Operations, Symbolics, Inc., Jun. 1992, pp. 15-60.

[12] S. Rowley, H. Shrobe, R. Cassels, "Joshua: Uniform Access to Heteregenous Knowledge Structures or Why Joshing is Better than Conniving or Planning," Symbolics Cambridge Research Center, Cambridge, MA.

4.3.1 Other related references

E. R. Tufte, The Visual Display of Quantitative Information, Cheshire, CT., Graphics Press ,1983.

P. Graham, Advanced Techniques for Common Lisp, Englewood Cliffs, NJ., Prentice-Hall, 1994.

5 Appendix

5.1 Code Listing

Joshua rules are defined in the file *rules.lisp*.

The user input function is defined in *user-input.lisp*.

The utility functions and other functions called by Joshua rules are defined in *utilities.lisp*.

Predicates which are used in the rules and resource knowledge (about resources in the environment), are defined in *rule-predicates.lisp*.

The definition of resource objects, as seen in Figure 3 is given in *resource-objects.lisp*.

Finally, a small sample of knowledge of available resources is given in *resource-knowledge.lisp*.

5.1.1 Rules.lisp

W:>GSDOLINA>thg2>rules.lisp.4

```
;;; -*- Mode: Joshua; Package: USER (really JOSHUA-USER); Syntax: Joshua -*-
;;; Created 5/02/99 16:25:30 by gsdolina running on POLK at MIT.

#||This file contains the complete set of rules for the reasoning part of the system.
Convention used here:  the name of rule is same as name of final predicate in the
conclusion of the rule, or the first conclusion if there are multiple conclusions
such as in an "and" statement.||#

;;;SYNCHRONOUS DISCUSSION RULES;;;


;Helper functions:
(defun avg (arg1 arg2)
  (/ (+ arg1 arg2) 2))


(defrule near (:backward)
  if [and [location-properties ?location1 ?floor1 ?room1 ?building1]
          [location-properties ?location2 ?floor2 ?room2 ?building2]
          (eql ?building1 ?building2)]
  then [near ?location1 ?location2])

(defrule far (:backward)
  if [and [location-properties ?location1 ?floor1 ?room1 ?building1]
          [location-properties ?location2 ?floor2 ?room2 ?building2]
          (not (eql ?building1 ?building2))]
  then [far ?location1 ?location2])

(defrule local-synch-possible (:backward)
  if [and [synchronous-possible ?user ?person-resource ?slot]
          [location-of ?user ?location1]
          [location-of ?person-resource ?location2]
          [near ?location1 ?location2]]
then [local-synch-possible ?user ?person-resource ?location2 ?slot])

(defrule has-access (:backward)
  if [and [location-of ?person ?location]
          [type-of ?person person]
          [location-of ?resource ?location]
          [or [type-of ?resource equipment]
              [type-of ?resource computer]]]
  then [has-access ?person ?resource])



(defrule service-supported (:backward)
  if [and [has-access ?person1 ?resource]
          [has-access ?person2 ?resource]
          [required ?resource ?service]]
  then [service-supported ?person1 ?person2 ?service])

(defrule experience (:backward)
  if [and [age ?person ?age]
          (unify ?experience (experience-from-age-f ?age))]
  then [experience ?person ?experience])

(defrule utility-from-experience (:backward)
  then [utility-from-experience ?person ?utility]
  if [and [experience ?person ?experience]
          (unify ?utility (utility-from-experience-f ?experience))])

;;;The overall utility is the sum of all the "specific" utilities,
;;;weighed by the utility criteria given by the user.

(defrule person-utility (:backward)
  if [and [utility-from-experience ?person ?u-experience]
          [utility-from-expertise ?person ?u-expertise]
          (unify ?overall (+ ?u-experience ?u-expertise))]
  then [person-utility ?person ?overall])

;Note the difference between experience and expertise.

;Utility from expertise depends on the various utility criteria.
;Some other utility parameters:  real-time, graphics,
;remote, multi-user, interactive, audio, private, speed.
;These apply to all the services/utilities, not
```

W:>GSDOLINA>thg2>rules.lisp.4

```
;just the person resource.

(defrule utility-from-expertise (:backward)
  if [and [topic-request ?topic]
          [expertise ?person-object ?topic ?level]      ;?level is 1 to 5
          (unify ?utility (utility-from-expertise-f ?level))]
  then [utility-from-expertise ?person-object ?utility])


(defrule utility-from-speed (:backward)
  then [utility-from-speed ?resource ?utility]
  if [and [speed-request ?request ?factor-r]
          [speed-factor-resource ?resource ?factor]
          (unify ?utility (utility-from-speed-f ?factor-r ?factor))])

(defrule busy-from-deadlines (:backward)
  if [and [deadline ?person ?deadline-list]
          (unify ?busy (float (busy-from-deadlines-f ?deadline-list)))]
  then [busy-from-deadlines ?person ?busy])   ;linear scale to 5

(defrule busy-overall (:backward)
  if [and [type-of ?person person]
          [busy-inherent ?person ?busy-p]
          [busy-from-deadlines ?person ?busy-d]
          (unify ?busy (/ (+ ?busy-p ?busy-d) 2))]
  then [busy-overall ?person ?busy]



;Rules that describe how busy other resources are:

(defrule cost-from-busy (:backward)
  if [busy-overall ?resource ?busy]
  then [cost-from-busy ?resource ?busy])

(defrule overall-cost (:backward)
  if [and [type-of ?resource person]
          [cost-from-busy ?resource ?cost1]
          [cost-inherent ?resource ?cost2]
          (unify ?total (+ ?cost1 ?cost2))]
  then [overall-cost ?resource ?total])

(defrule cost-inherent1 (:backward)
  if [and [type-of ?resource person]
          [experience ?resource ?experience]]
  then [cost-inherent ?resource ?experience])

(defrule speed-from-busy (:backward)
  if [and [or [type-of ?resource equipment]
              [type-of ?resource hardware]]
          [busy-overall ?resource ?busy] ;between 1 and 6.
          (unify ?factor (- 6 ?busy))]
  then [speed-from-busy ?resource ?factor])

(defrule speed-factor-resource (:backward)
  if [and [speed-from-busy ?resource ?factor]
          [speed-inherent ?resource ?factor2]
          (unify ?overall (avg ?factor ?factor2))]
  then [speed-factor-resource ?resource ?overall])


;;;Now some rules which will enable a discussion at a given time;;;;

(defrule synchronous-possible (:backward)
  if [and [free-time ?person ?time-list]
          [service-request ?user ■]
          [time-request ?time-slot ?factor]
          (unify ?time-granted (fit-into-time-list? ?time-slot ?time-list ?factor))
          (not (null? ?time-granted))]
  then [synchronous-possible ?user ?person ?time-granted])


(defrule synchronous-not-possible (:backward)
  if [and [free-time ?person ?time-list]
          [person-request ?person]
          [service-request ?user ■]
```

```
          [time-request ?time-slot ?factor]
          (unify ?time-granted (fit-into-time-list? ?time-slot ?time-list ?factor))
          (null? ?time-granted)
          (progn
            (tell [synch-not-possible2 ?user ?person]))]
  then [synchronous-not-possible ?user ?person])

(defrule answer-synch-discussion (:backward)
  if [and [service-request ?user =]
          [best-person-time ?person ?time]
          [best-service-place-resource ?service ?place ?resource]]
  then [answer-synch-discussion ?user ?person ?service ?resource ?time ?place])

(defrule best-person1 (:backward :importance 200)
  if [and [person-request ?person]
          [service-request ?user =]
          [synchronous-possible ?user ?person ?time]
          (progn
            (tell [best-person-time ?person ?time])
            (untell [person-chosen-already ?truth])
            (tell [person-chosen-already t])
            (tell [chosen-person ?person])
                  (succeed))]
  then  [best-person-time ?person ?time])


(defrule possible-person (:backward)
  if [and [topic-request ?topic]
          [expertise ?person ?topic ?level]
          [person-request ?requested-person]
          [service-request ?user =]
          [synchronous-not-possible ?user ?requested-person]
          [synchronous-possible ?user ?person ?time]
          (progn
            (tell [possible-person ?person ?time])
            (succeed))]
  then [possible-person ?person ?time])

;;;Now we have several possible people.

(defrule utility/cost (:backward)
  if [and [person-utility ?person ?utility]
          [overall-cost ?person ?cost]
          (unify ?ratio (/ ?utility ?cost))
          (progn
            (tell [person-u/c ?person ?ratio])
            (succeed))]
  then [person-u/c ?person ?ratio])

(defrule compared (:forward)          ;This is FORWARD.
  if [and [possible-person ?person =] ;the goal is the side-effect
          [person-u/c ?person ?ratio]
          (unify ?side-effect (utility-maximizer ?ratio))]
  then [compared ?person])

(defrule best-person2 (:backward :importance 150)
  if [and [person-chosen-already nil]
          [possible-person ?person ?time]
          [person-u/c ?person ?ratio]
          [compared ?person]
          (unify ?truth (max-utility? ?ratio))
          ?truth
          (progn
            (tell [best-person-time ?person ?time])
            (untell [person-chosen-already ?truth])
            (tell [person-chosen-already t])
            (tell [chosen-person ?person])
                  (succeed))]
  then [best-person-time ?person ?time])

(defrule best-person3 (:backward :importance 100)
  if [and [person-request ?person]
          [service-request ?user ?service]
          [synch-not-possible2 ?user ?person]
          [topic-request ?topic]
          [expertise ?other ?topic ?level]
```

W:>GSDOLINA>thg2>rules.lisp.4

```
          [synchronous-possible ?user ?other ?time]]
  then [best-person-time ?other ?time])

;;;Now to find the best place

(defrule best-service-place-resource (:backward :importance 20)
  if [and [service-request ?user ■]
          [chosen-person ?person]
          [location-of ?user ?location1]
          [location-of ?person ?location2]
          [near ?location1 ?location2]]
  then [best-service-place-resource meeting ?location2 nil])

(defrule best-service-place-resource2 (:backward :importance 19)
  if [and [service-request ?user ■]
          [chosen-person ?person]
          [location-of ?user ?location1]
          [location-of ?person ?location2]
          [far ?location1 ?location2]
          [service-supported ?person ?user videoconference]]
  then [best-service-place-resource videoconference ?location1 nil])

(defrule best-service-place-resource3 (:backward :importance 18)
  if [and [service-request ?user ■]
          [chosen-person ?person]
          [location-of ?user ?location1]
          [location-of ?person ?location2]
          [far ?location1 ?location2]
          [phone ?location2 ?number]]
  then [best-service-place-resource phone ?location1 ?number])

;;;Now the asynchronous discussion:

(defrule answer-asynch-discussion (:backward)
  if [and [person-request ?person]
          [service-request ?user ■]
          [best-asynch-service ?output]
          (unify ?service (car ?output))
          (unify ?resource (cdr ?output))]
  then [answer-asynch-discussion ?user ?person ?service ?resource])

(defrule best-asynch-service (:backward)
  if [and [graphics-request ?number ■]
          [person-request ?person]
          [person-fax ?person ?fax]
          [email-address ?person ?address]
          (unify ?output (best-asynch-service-f ?number ?address ?fax))]
  then [best-asynch-service ?output])

(defun best-asynch-service-f (number address fax)
  (cond ((<= number 2)
         (cons 'email address))
        (t (cons 'fax fax))))

(defrule person-fax (:backward)
  if [and [type-of ?person person]
          [location-of ?person ?location1]
          [type-of ?resoure fax]
          [location-of ?resource ?location2]
          [near ?location1 ?location2]
          [fax-number ?resource ?fax]]
  then [person-fax ?person ?fax])


;;;These rules deal with Location

(defrule answer-location-service (:backward)
  if [and [person-request ?person]
          [location-of ?person ?location]
          (unify ?result (locator-f ?location))]
  then [answer-location-service ?person ?result])

(defun locator-f (location)
  ;(let ((now (get-universal-time)))
  (multiple-value-bind (secs minutes hours day month year dow) (get-decoded-time)
  (if (and (< dow 5)
```

```
          (< 7 hours)
          (< hours 17))
      location
      'not_available)))

;;;These rules deal with Printing

(defrule answer-print-service (:backward)
  if (and (service-request ?user =)
          (best-computer-printer ?computer ?printer))
  then (answer-print-service ?user ?computer ?printer))

(defrule best-computer-printer (:backward)
  if (and (printer-u/c ?printer ?ratio)
          (possible-computer-printer ?computer ?printer)
          (maximum-u/c ?ratio))
  then (best-computer-printer ?computer ?printer))

(defrule possible-computer-printer (:backward)
  if (and (service-request ?user =)
          (type-of ?resource computer)
          (has-access ?user ?resource)
          (can-print ?resource ?printer))
  then (possible-computer-printer ?resource ?printer))

(defrule printer-u/c (:backward)
  if (and (printer-utility ?printer ?utility)
          (printer-cost ?printer ?cost)
          (unify ?ratio (/ ?utility ?cost))
          (progn
            (maximizer ?ratio)))
  then (printer-u/c ?printer ?ratio))


(defun maximizer (number)
  (if (< max1 number)
      (progn
        (setq max1 number)
        (untell (maximum-u/c ?ratio))
        (tell `(maximum-u/c ,number)))))

(defrule printer-utility (:backward)
  if (and (possible-computer-printer = ?printer)
          (printer-properties ?printer ?resolution ?color ?speed)
          (resolution-request ?res1)
          (speed-request ?spd1 =)
          (color-request ?col1 ?colf)
          (close-to-request ?person ?distance)
          (near2 ?person ?printer ?factor)
          (unify
            ?utility (printer-utility-f ?resolution ?color ?speed ?res1 ?spd1
                                        ?col1 ?distance ?factor)))
  then (printer-utility ?printer ?utility))

(defun printer-utility-f (pres pcol pspd res spd col pdist dist)
  (+ (* pres res)
     (* pcol col)
     (* pspd spd)
     (* dist (- 0 pdist))))


(defrule near2 (:backward)
  if (and (location-of ?resource1 ?loc1)
          (location-properties ?loc1 ?floor1 ?room1 ?blg1)
          (location-of ?resource2 ?loc2)
          (location-properties ?loc2 ?floor2 ?room2 ?blg2)
          (unify ?factor (near2-f ?floor1 ?room1 ?blg1 ?floor2 ?room2 ?blg2)))
  then (near2 ?resource1 ?resource2 ?factor))

(defun near2-f (floor1 room1 blg1 floor2 room2 blg2)
  (let ((floors (+ 1 (abs (- floor1 floor2))))
        (rooms (abs (- room1 room2))))
    (if (eql blg1 blg2)
        (* floors rooms)
        500))) ;a big number. In the sample environment,
;everything is in the same building.
```

W:>GSDOLINA>thg2>rules.lisp.4

```
;;;Notification rules

(defrule answer-notification1 (:backward)
  if (and (urgency-request ?urgency)
          (< 4 ?urgency)
          (person-request ?person)
          (has-stuff ?person ?thing ?number)
          (or (eql ?thing 'pager)
              (eql ?thing 'cellular)))
  then (answer-notification ?person ?thing ?number))

(defrule answer-notification2 (:backward)
  if (and (urgency-request ?urgency)
          (and (<= ?urgency 4)
               (> ?urgency 2))
          (person-request ?person)
          (location-of ?person ?location)
          (phone ?location ?phone))
  then (answer-notification ?person telphone ?phone))

(defrule answer-notification3 (:backward)
  if (and (urgency-request ?urgency)
          (<= ?urgency 4)
          (person-request ?person)
          (email-address ?person ?address))
  then (answer-notification ?person email ?address))
```

5.1.2 User-input.lisp

W:>GSDOLINA>thg2>user-input.lisp.5

```
;;; -*- Mode: Joshua; Package: USER (really JOSHUA-USER); Syntax: Joshua -*-
;;; Created 5/01/99 19:00:43 by gsdolina running on POLK at MIT.

;;;Set lips context to Joshua:
;(cp:execute-command "set lisp context" "joshua")
;;;Here are some helper functions:

(defvar person-list '(gsdolina hes rladdaga blumberg cvince gregs jcma rhu reti))
(defvar service-list '(discussion info-access location-service print-service
                                    notification give-info))
(defvar topics '(brainstorming hacking lisp management money politics thesis))
(defvar topic-list (append topics '(other)))
(defun round-time (time)
  (if (eql time nil)
      nil
      (multiple-value-bind (secs minutes hours day month year)
          (time:decode-universal-time time)
        (declare (ignore secs))
        (setq minutes (* 10 (round minutes 10)))
        (time:encode-universal-time 0 minutes  hours day month year))))


(defun null? (arg) (eql arg nil))
(defun filter (element l)
  (cond ((null? l) l)
        ((eql element (car l)) (filter element (cdr l)))
        (t (append (list (car l)) (filter element (cdr l))))))

;(accept '((dw:member-sequence ,person-list)))
;;;Here are some lists of options which the user will choose:
;;I can also have a topic-list.


;;Here are the predicates which hold the user's input information:
(define-predicate service-request (user type))
(define-predicate discussion-request (discussion-type))
(define-predicate time-request (time-slot time-f))
(define-predicate topic-request (topic))
(define-predicate person-request (whom))
(define-predicate graphics-request (graphics graphics-f))
(define-predicate speed-request (speed speed-f))
(define-predicate resolution-request (resolution))
(define-predicate color-request (color color-r))
(define-predicate close-to-request (close-to distance-f))
(define-predicate urgency-request (urgency))

(define-predicate answer-synch-discussion (user person service resource time place))
(define-predicate answer-asynch-discussion (user person service resource))
(define-predicate answer-info-access (user resource topic))      ;This is only tentative
(define-predicate answer-location-service (person location))
(define-predicate answer-print-service (user computer printer))
(define-predicate answer-notification (person service resource))
(define-predicate answer-give-info (user person service-resource-list))



;;Now the main interface function which gathers information from the user:

(defun user-input (&key (stream *standard-output*) pop? )

  ;;First I clear the old predications from the Joshua database
  (untell [service-request ?user ?type])
  (untell [discussion-request ?discussion-type])
  (untell [time-request ?time-slot ?time-f])
  (untell [topic-request ?topic])
  (untell [person-request ?whom])
  (untell [graphics-request ?graphics ?graphics-f])
  (untell [speed-request ?speed ?speed-f])
  (untell [resolution-request ?resolution])
  (untell [color-request ?color ?color-f])
  (untell [close-to-request ?close-to ?distance-f])
  (untell [urgency-request ?urgency])
  (untell [chosen-person ?person])
  (untell [chosen-time ?time])
  (untell [best-service-place-resource ?service ?place ?resource])
  (untell [person-chosen-already ?truth])
```

W:>GSDOLINA>thg2>user-input.lisp.5

```
(untell [possible-person ?person ?time])
(untell [person-u/c ?person ?ratio])
(untell [best-person-time ?person ?time])
(untell [compared ?person])
(untell [synch-not-possible2 ?user ?person])
(setq max1 0)

(let ((user nil)(type nil)(time-slot nil)(topic nil)(speed 0)(graphics 0)
      (resolution 5)(color 0)(discussion-type nil)(whom nil)(urgency nil)(close-to nil)
      (time-f 6)(color-f 0)
      (speed-f 0)(graphics-f 0)(distance-f 0)(other-topic nil))

  (dw:accepting-values (stream :own-window pop? :label "Please answer these questions")
    (setq user  (accept `((dw:member-sequence ,person-list))
                        :default user :stream stream :prompt "Who are you")
          type  (accept `((dw:member-sequence ,service-list))
                        :default type :stream stream :prompt "Choose a service"))
    (cond ((eql type 'discussion)
           (setq whom (accept `((dw:member-sequence ,(filter user person-list)))
                              :default whom :stream stream :prompt "With whom")
                 topic (accept `((dw:member-sequence ,topic-list))
                               :default topic :stream stream :prompt "Topic"))
           (if (eql topic 'other)
               (setq other-topic (accept 'symbol :default other-topic
                                         :stream stream :prompt "Other")))

           (setq discussion-type (accept '(member synchronous asynchronous)
                                         :default discussion-type :stream stream
                                         :prompt "Discussion type"))
           (if (eql discussion-type 'synchronous)
               (setq time-slot (cons (accept '(time:universal-time) :stream stream
                                             :prompt "when")
                                     (accept '(time:time-interval)
                                             :stream stream :prompt "for how long"))
                     time-f (accept '(member 0 1 2 3 4 5 6) :default time-f
                                    :stream stream :prompt "time factor"))
               (if (eql discussion-type 'asynchronous)
                   (setq graphics (accept '(member 0 1 2 3 4)
                                          :default graphics :stream stream
                                          :prompt "Need graphics")
                         graphics-f graphics))))
          ((eql type 'info-access)
           (setq topic (accept `((dw:member-sequence ,topic-list))
                               :default topic :stream stream :prompt "Topic"))
           (if (eql topic 'other)
               (setq other-topic (accept 'symbol :default other-topic
                                         :stream stream :prompt "Other")))
           (setq speed (accept '(member 0 1 2 3 4 5 6) :default speed
                               :stream stream :prompt "Speed")
                 speed-f speed))


          ((eql type 'location-service)
           (setq whom (accept `((dw:member-sequence ,(filter user person-list)))
                              :default whom
                              :stream stream
                              :prompt "Who are you looking for")))

          ((eql type 'print-service)
           (setq resolution (accept '(member 0 1 2 3 4 5 6)
                                    :default resolution :stream stream
                                    :prompt "Resolution")
                 color (accept '(member 0 1 2 3 4 5 6)
                               :default color :stream stream :prompt "Color mode")
                 color-f color
                 speed (accept '(member 0 1 2 3 4 5 6) :default speed
                               :stream stream :prompt "Speed")
                 speed-f speed
                 close-to (accept `((dw:member-sequence ,person-list))
                                  :default user :stream stream :prompt "Print close to")
                 distance-f (accept '(member 0 1 2 3 4 5 6)
                                    :default distance-f :stream stream
                                    :prompt "distance factor")))

          ((eql type 'notification)
           (setq whom (accept `((dw:member-sequence ,(filter user person-list)))
```

W:>GSDOLINA>thg2>user-input.lisp.5

```
                                :default whom :stream stream
                                :prompt "Whom would you like to notify")
                        urgency (accept '(member 0 1 2 3 4 5 6)
                                        :default urgency :stream stream :prompt
                                        "Choose urgency")))))

    ;here I pass the user info to joshua

    (tell `[service-request ,user ,type])
    (tell `[discussion-request ,discussion-type])

    (tell `[time-request ,time-slot ,time-f])
    (if (eql topic 'other)
        (tell `[topic-request ,other-topic])
        (tell `[topic-request ,topic]))
    (tell `[person-request ,whom])
    (tell `[graphics-request ,graphics ,graphics-f])
    (tell `[speed-request ,speed ,speed-f])
    (tell `[resolution-request ,resolution])
    (tell `[color-request ,color ,color-f])
    (tell `[close-to-request ,close-to ,distance-f])
    (tell `[urgency-request ,urgency])
;Some standard initializations:
    (tell [person-chosen-already nil])


    ;Now I ask the appropriate question.

    (cond ((and (eql type 'discussion)(eql discussion-type 'synchronous))
           (ask [answer-synch-discussion ?user ?person ?service ?resource ?time ?place]
                #'say-query))
          ((and (eql type 'discussion)(eql discussion-type 'asynchronous))
           (ask [answer-asynch-discussion ?user ?person ?service ?resource]
                #'say-query))
          ((eql type 'info-access)
           "Info-access is currently not supported.  Please try a discussion instead!" )
          ((eql type 'location-service)
           (ask [answer-location-service ?person ?location]
                #'say-query))
          ((eql type 'print-service)
           (ask [answer-print-service ?user ?computer ?printer]
                #'say-query))
          ((eql type 'notification)
           (ask [answer-notification ?person ?service ?resource]
                #'say-query))
          ((eql type 'give-info)
           (ask [answer-give-info ?user ?person ?s-r-list]
                #'print-query)))))



;;The following methods format the output in an understandable way:

(define-predicate-method (say answer-synch-discussion) (&optional (stream *standard-output*))
  (with-statement-destructured (user person service resource time place) self
    (format stream
            "~& You can have ~A with ~A in ~A between ~\\time\\  and ~\\time\\ using ~A"
            service person place (car time) (cdr time) resource)))

(define-predicate-method (say answer-asynch-discussion) (&optional (stream *standard-output*))
  (with-statement-destructured (user person service resource) self
    (format stream
            "~& You can ~A the person ~A at ~A"
            service person resource)))

(define-predicate-method (say answer-location-service) (&optional (stream *standard-output*))
  (with-statement-destructured (person location) self
    (format stream
            "~& The location of ~A is ~A " person location)))

(define-predicate-method (say answer-print-service) (&optional (stream *standard-output*))
  (with-statement-destructured (user computer printer) self
    (format stream
            "~& You can use the computer ~A to print to ~A " computer printer)))

(define-predicate-method (say answer-notification) (&optional (stream *standard-output*))
```

W:>GSDOLINA>thg2>user-input.lisp.5

```
(with-statement-destructured (person service resource) self
  (format stream
          "~& You can notify ~A with ~A using ~A " person service resource)))
```

5.1.3 Utilities.lisp

W:>GSDOLINA>thg2>utilities.lisp.2

```
;;; -*- Mode: Joshua; Package: USER (really JOSHUA-USER); Syntax: Joshua -*-
;;; Created 5/01/99 21:25:32 by gsdolina running on POLK at MIT.

;;;This file contains general utilities for working with the knowledge

;;; This function adds a user into the list of users, called person-list

(defun add-user (user) (push user person-list))
(defun add-topic (topic) (pushnew topic topics))
;;; This function removes a user from the list of known users

(defun remove-user (user)
  (setq person-list (delete user person-list)))

(defun remove-topic (topic)
  (setq topic-list (delete topic topic-list)))

;;;Time manipulations
;To print out time, use the presentation type below:
;      (present 3134410800 '((time:universal-time) :long-date t))

;I like this short format better:
;(present 3134410800 '((time:universal-time)))

;I can also do it like this:
;(present 3134410800 'time:universal-time)



(defun interval (time1 time2)
  (time:print-interval-or-never (abs (- time1 time2))))

;For a free-slot will use a cons pair of a start-time and end-time
;(cons start-time end-time)  Given this and a request as input, I need a function which
;will check if the request is satisfiable with the available time.  The requested-time
;is a cons pair of start-time and duration (interval).
;Time and interval are both universal-time.

(defun between? (argument start end)
  (and (>= argument start)
       (>= end argument)))

(defun flexibility (factor)
  (cond ((= factor 6) 0)                          ;no flexibility
        ((= factor 5) 1800)                       ;30 minutes
        ((= factor 4) 3600)                       ;1 hour
        ((= factor 3) 14400)                      ;4 hours
        ((= factor 2) 86400)                      ;1 day
        ((= factor 1) (* 7 86400))                ;1 week
        ((= factor 0) (* 4 (* 7 86400)))))        ;1 month

(defun fit-in-time-slot? (requested-time free-slot factor)
  (let* ((margin (flexibility factor))
         (f-start (- (car free-slot) margin))
         (f-end (+ (cdr free-slot) margin))
         (r-start (car requested-time))
         (r-end (+ r-start (cdr requested-time))))

    (if (and (between? r-start f-start f-end)
             (between? r-end f-start f-end))
        free-slot
        'fail)))

;The funcion below maps fit-in-time-slot? over a time-list of free-slots, and returns t if
;the requested time fits into one of the slots, otherwise nil.

(defun fit-into-time-list? (request time-list factor)
  (let ((return-list
          (map 'list (lambda(arg)(fit-in-time-slot? request arg factor)) time-list)))
    (car (delete 'fail return-list))))          ;this returns the possible slot or nil

;;;     (not (atom (zl:memq 't truth-list)))))

;;;This function will take a requested time slot and if it fits into a free time slot,
;;;take that time out of the free time slot, and return the new free time slot or slots.
```

W:>GSDOLINA>thg2>utilities.lisp.2

```
(defun assign-time (requested-time free-slot)
  (let* ((f-start (car free-slot))
         (f-end (cdr free-slot))
         (r-start (car requested-time))
         (r-end (+ r-start (cdr requested-time))))
    (if (and (between? r-start f-start f-end)
             (between? r-end f-start f-end))
        "Take the time out of the free slot and return one or two new free time slots"
        "Sorry, but the requested time doesn't fit in the free time.")))

;;;Experience

(defun experience-from-age-f (age)
  (max (- age 20) 0))

(defun utility-from-experience-f (years)
  (cond ((<= years 10) (/ (zl:^ years 2) 2))
        ((and (< 10 years)
              (<= years 13))
         (+ (utility-from-experience-f 10)
            (+ (* 10 (- years 10))(* -1.5 (zl:^ (- years 10) 2)))))
        (t (+ (utility-from-experience-f 13)
              (+ (- years 13) (* -.01569 (zl:^ (- years 13) 2)))))))

;;;Expertise
(defun utility-from-expertise-f (level)
  (* 82 (/ level 5)))

;;;Speed
(defun utility-from-speed-f (user-factor resource-speed-factor)
  (* user-factor resource-speed-factor))

;;;Here are the functions used by the cost, busy, and utility rules.
(defun busy-utility-helper (deadline)
  (let ((now (get-universal-time))
        (2weeks 1209600)
        (deadline-time (car deadline))
        (importance (cdr deadline)))
    (cond ((<= deadline-time now) 0)
          ((>= (* importance 2weeks) (- deadline-time now))
           (float (* 5 (/ (- now (- deadline-time (* importance 2weeks)))
                          (* importance 2weeks)))))
          (t 0))))

(defun accumulate (lst)
  (if (null? lst)
      0
      (+ (car lst) (accumulate (cdr lst)))))

(defun busy-from-deadlines-f (deadline-list)
  (let ((list-of-values (map 'list (lambda (arg)(busy-utility-helper arg)) deadline-list)))
    (accumulate list-of-values)))

;;;More on utility

(defvar max-utility 0)
(defun utility-maximizer (arg)
  (if (> arg max-utility)
      (setq max-utility arg)))

(defun max-utility? (utility)
  (>= utility max-utility))
```

5.1.4 Rule-predicates.lisp

W:>GSDOLINA>thg2>rule-predicates.lisp.4

```
;;; -*- Mode: Joshua; Package: USER (really JOSHUA-USER); Syntax: Joshua -*-
;;; Created 5/05/99 12:32:03 by gsdolina running on SOUFRIERE-VLM at MIT.

(define-predicate location-properties (location floor room building))
(define-predicate location-of (resource location))
(define-predicate near (location1 location2))
(define-predicate type-of (resource type))
(define-predicate has-access (person resource))
(define-predicate required (resources service))
(define-predicate service-supported (user person service))
(define-predicate age (person age))
(define-predicate experience (person years))
(define-predicate utility-from-experience (person utility))
(define-predicate utility-from-expertise (person utility))
(define-predicate utility-from-speed (person utility))
(define-predicate person-utility (person utility))      ;overall utility
(define-predicate expertise (person topic level))
(define-predicate speed-factor-resource (resource factor))
(define-predicate busy-from-deadlines (person busy))
(define-predicate deadline (person deadline-list))
(define-predicate current-time (time))
(define-predicate busy-inherent (person busy))
(define-predicate busy-overall (resource busy))
(define-predicate cost-from-busy (resource cost))
(define-predicate cost-inherent (resource cost))
(define-predicate cost-from-desirability (resource cost))
(define-predicate overall-cost (resource cost))
(define-predicate speed-from-busy (resource speed))
(define-predicate speed-inherent (resource speed))
(define-predicate free-time (person time-list))
(define-predicate synchronous-possible (user person time-slot))
(define-predicate local-synch-possible (user person loction time-slot))
(define-predicate synchronous-not-possible (user person))
(define-predicate possible-person (person time))
(define-predicate best-person-time (person time))
(define-predicate compared (person))
(define-predicate person-u/c (person ratio))
(define-predicate best-place (location))
(define-predicate best-service (service resource))
(define-predicate service-class (class))
(define-predicate far (location1 location2))
(define-predicate chosen-person (person))
(define-predicate chosen-time (time))
(define-predicate best-service-place-resource (service place resource))
(define-predicate phone (office number))
(define-predicate person-chosen-already (truth))
(define-predicate synch-not-possible2 (user person))
(define-predicate best-asynch-service (output))
(define-predicate email-address (person address))
(define-predicate person-fax (person number))
(define-predicate fax-number (fax-machine number))
(define-predicate can-print (computer printer))
(define-predicate near2 (resource1 resource2 factor))
(define-predicate printer-properties (printer resolution color speed))
(define-predicate printer-cost (printer cost))
(define-predicate printer-utility (printer utility))
(define-predicate printer-u/c (printer ratio))
(define-predicate best-computer-printer (computer printer))
(define-predicate possible-computer-printer (computer printer))
(define-predicate maximum-u/c (ratio))
(define-predicate has-stuff (person thing info))
```

5.1.5 Resource-objects.lisp

W:>GSDOLINA>thg2>resource-objects.lisp.3

```
;;; -*- Mode: Joshua; Package: USER (really JOSHUA-USER); Syntax: Joshua -*-
;;; Created 4/20/99 15:30:01 by gsdolina running on POLK at MIT.
;This file is a proposed, new representation of resources as Joshua objects.
;It is a more natural and logical representation than the regular predicates,
;while Joshua can perform all of it's reasoning about the objects equally well.

(define-object-type resource
  :slots (uniqe-id
          (cost :intiform 0 :attached-actions)
          (busy :initform NIL)))


(define-object-type date-object
  :slots (month day year))

(define-object-type address-object
  :slots (number street apt city state zip-code))

(define-object-type phone-object
  :slots (number area-code local-number extension (conference :initform 1)))

;The "number" slot holds 617-225-9157x34, while "local-number" holds 2259157.  This
;is useful for the area-code accessor, so I can check distances of people,
;and also determine the cost of long distance phone calls.

(define-object-type pager-object
  :slots (number area-code local-number pin))

;The "number" slot holds the whole number with everything, and "local-number"
;holds only the local number

(define-object-type building-object
  :slots (name number)
  :parts ((address address-object)))

(define-object-type schedule-object
  :slots ((free :set-valued t)
          (deadline :set-valued t)))

(define-object-type location-object
  :slots (name description location floor-number capacity desks chairs board)
  :parts ((building building-object)
          (phone phone-object)
          (schedule schedule-object))
  :included-object-types (resource))

(make-object 'location-object :name 'ne43-803)
(make-object 'location-object :name 'location-foo)

(tell [value-of (ne43-803 name) location-foo])

(define-object-type software-object
  :slots ((video-conference :set-valued t)
          (communication :set-valued t)
          (email :set-valued t)
          (web :set-valued t))) ;these slots will hold the list of various software
                                ;packages organized by these categories

(define-object-type peripherals-object
  :slots (microphone audio (video :set-valued t)))

(define-object-type os-object
  :slots (name full-name maker version compatible
                (attributes :set-valued t)))

(define-object-type monitor-object
  :slots (resolution color model))
;The "color" will hold either bw, true-color, etc. (see MGA display properties on PC)
;The "resolution" is low, medium, high

(define-object-type computer-object
  :slots (name type (printer :set-valued t))
  :parts ((os os-object)
          (monitor monitor-object)
          (software software-object)
          (location location-object)
```

W:>GSDOLINA>thg2>resource-objects.lisp.3

```
          (peripherals peripherals-object))
  :included-object-types (resource))



(define-object-type person-object
  :slots (first-name middle-initial last-name email age money
                     (expertise-3 :set-valued t)
                     (expertise-2 :set-valued t)
                     (expertise-1 :set-valued t)
                     (personality-3 :set-valued t)
                     (personality-2 :set-valued t)
                     (personality-1 :set-valued t))
  :parts ((birthday date-object)
          (location location-object)
          (schedule schedule-object)
          (work-phone phone-object)
          (home-phone phone-object)
          (cell-phone phone-object)
          (home-address address-object)
          (pager pager-object))
  :included-object-types (resource))


(define-object-type printer-object
  :slots (name model speed delay (p-format :set-valued t) host)
  :parts ((location location-object))
  :included-object-types (resource))
```

5.1.6 Resource-knowledge.lisp

W:>GSDOLINA>thg2>resource-knowledge.lisp.4

```
;;; -*- Mode: Joshua; Package: USER (really JOSHUA-USER); Syntax: Joshua -*-
;;; Created 5/07/99 18:42:03 by gsdolina running on SOUFRIERE-VLM at MIT.

;;;This is sample resource knowledge used in the system.


;Some deadline assignemnts:

(setq deadline-list1 '((3135600000 . 3) (3135603600 . 2) (3136284000 . 5)))
(setq deadline-list2 '((3136248000 . 2) (3137025600 . 3)))
(setq deadline-list3 '((3139740000 . 3) (3140049600 . 5)))
(setq deadline-list4 '((3139704000 . 2)))
(setq deadline-list5 '((3136248000 . 5) (3137976000 . 3)))
(setq deadline-list6 '((3139704000 . 3)))
(setq deadline-list7 '((3140049600 . 4) (3143160000 . 2)))
(setq deadline-list8 '((3137061600 . 1)))
(setq deadline-list9 '((3137061600 . 1)))

;;;Free times:  these shoud be entered automatically.
(defun getin ()
  (setq free-list (list (cons (accept 'time:universal-time)
                              (accept 'time:universal-time))
                        (cons (accept 'time:universal-time)
                              (accept 'time:universal-time)))))

(setq free-list1 '((3135333600 . 3135337200) (3135340800 . 3135346200)
                   (3135351600 . 3135358800) (3135423600 . 3135438000)))
(setq free-list2 '((3135344400 . 3135351600) (3135423600 . 3135425400)
                   (3135506400 . 3135517200) (3135520800 . 3135526200)))
(setq free-list3 '((3135337200 . 3135340800) (3135344400 . 3135351600)
                   (3135355200 . 3135358800) (3135423600 . 3135427200)))
(setq free-list4 '((3135333600 . 3135340800) (3135420000 . 3135423600)
                   (3135434400 . 3135445200) (3135506400 . 3135513600)))
(setq free-list5 '((3135337200 . 3135340800) (3135344400 . 3135351600)
                   (3135355200 . 3135358800) (3135420000 . 3135427200)))
(setq free-list6 '((3135333600 . 3135344400) (3135348000 . 3135351600)
                   (3135355200 . 3135358800) (3135416400 . 3135423600)))
(setq free-list7 '((3135340800 . 3135358800) (3135445200 . 3135452400)
                   (3135510000 . 3135524400) (3135531600 . 3135542400)))
(setq free-list8 '((3135441600 . 3135445200)))
(setq free-list9 '((3135510000 . 3135524400) (3135600000 . 3135610800)))

(tell [and [location-of gsdolina ne43-803]
           [type-of gsdolina person]
           [age gsdolina 22]
           [expertise gsdolina europe 3]
           [expertise gsdolina thesis 3]
           [expertise gsdolina lisp 1]
           [expertise gsdolina skiing 4]
           [expertise gsdolina joshua 2]
           `[deadline gsdolina ,deadline-list1]
           [busy-inherent gsdolina 2]
           [cost-inherent gsdolina 2]
           [speed-inherent gsdolina 3]
           `[free-time gsdolina ,free-list1]])


(tell [and [location-of hes ne43-839]
           [type-of hes person]
           [age hes 52]
           [expertise hes lisp 5]
           [expertise hes thesis 4]
           [expertise hes joshua 5]
           [expertise hes money 4]              ;These would typically be self-descriptions.
           `[deadline hes ,deadline-list2]
           [busy-inherent hes 5]
           [cost-inherent hes 4]
           [speed-inherent hes 3]
           `[free-time hes ,free-list2]])

(tell [and [location-of rladdaga ne43-804]
           [type-of rladdaga person]
           [age rladdaga 49]                    ;Unknown... safe estimate.
           [expertise rladdaga lisp 4]
           [expertise rladdaga money 5]
           [expertise rladdaga politics 4]
```

```
          [expertise rladdaga management 4]
          `[deadline rladdaga ,deadline-list3]
          [busy-inherent rladdaga 3]
          [cost-inherent rladdaga 3]
          [speed-inherent rladdaga 4]
          `[free-time rladdaga ,free-list3]])

(tell [and [location-of blumberg ne43-803]
          [type-of blumberg person]
          [age blumberg 22]
          [expertise blumberg lisp 5]
          [expertise blumberg MacIvory 3]
          [expertise blumberg joshua 2]
          [expertise blumberg hacking 3]
          `[deadline blumberg ,deadline-list4]
          [busy-inherent blumberg 2]
          [cost-inherent blumberg 2]
          [speed-inherent blumberg 3]
          `[free-time blumberg ,free-list4]])

(tell [location-of cvince ne43-832])
(tell [type-of cvince person])
(tell [age cvince 22])
(tell      [expertise cvince lisp 4])
(tell      [expertise cvince thesis 2])
(tell      [expertise cvince VLM 4])
(tell      [expertise cvince java 4])
(tell      `[deadline cvince ,deadline-list5])
(tell      [busy-inherent cvince 2])
(tell      [cost-inherent cvince 2])
(tell      [speed-inherent cvince 3])
(tell      `[free-time cvince ,free-list5])

(tell [and [location-of gregs ne43-802]
          [type-of gregs person]
          [age gregs 35]                          ;Again a rough estimate
          [expertise gregs java 3]
          [expertise gregs politics 3]
          [expertise gregs thesis 1]
          [expertise gregs money 3]
          `[deadline gregs ,deadline-list6]
          [busy-inherent gregs 2]
          [cost-inherent gregs 2]
          [speed-inherent gregs 3]
          `[free-time gregs ,free-list6]])

(tell [and [location-of jcma ne43-807]
          [type-of jcma person]
          [age jcma 35]                           ;Again a guess...
          [expertise jcma lisp 5]
          [expertise jcma hacking 5]
          [expertise jcma thesis 3]
          [expertise jcma brainstorming 4]
          [expertise jcma rock-climbing 5]
          `[deadline jcma ,deadline-list7]
          [busy-inherent jcma 4]
          [cost-inherent jcma 4]
          [speed-inherent jcma 5]
          `[free-time jcma ,free-list7]])

(tell [and [location-of rhu ne43-807]
          [type-of rhu person]
          [age rhu 45]                            ;estimate
          [expertise rhu politics 5]
          [expertise rhu money 4]
          [expertise rhu infostructure 5]
          `[deadline rhu ,deadline-list8]
          [busy-inherent rhu 2]
          [cost-inherent rhu 2]
          [speed-inherent rhu 3]
          `[free-time rhu ,free-list8]])

(tell [and [location-of reti ne43-703]
          [type-of reti person]
          [age reti 40]                                   ;estimate
          [expertise reti lisp 5]
```

```
           [expertise reti hacking 5]
           [expertise reti MacIvory 5]
           [expertise reti VLM 5]
           `[deadline reti ,deadline-list9]
           [busy-inherent reti 4]
           [cost-inherent reti 4]
           [speed-inherent reti 5]
           `[free-time reti ,free-list9]]))
(tell [and [phone ne43-802 617-253-xxxx]
           [phone ne43-803 617-253-6567]
           [phone ne43-807 617-253-5966]
           [phone ne43-839 617-253-7877]
           [phone ne43-703 617-253-xxxx]
           [phone ne43-832 617-253-5043]]))

(tell [and [location-properties ne43-803 8 803 ne43]
           [location-properties ne43-804 8 804 ne43]
           [location-properties ne43-802 8 802 ne43]
           [location-properties ne43-839 8 839 ne43]
           [location-properties ne43-832 8 832 ne43]
           [location-properties ne43-807 8 807 ne43]
           [location-properties ne43-703 8 703 ne43]
           [location-properties ne43-830 8 830 ne43]])

(tell [and [location-of gregs ne43-802]
           [type-of gregs person]
           [age gregs 35]                          ;Again a rough estimate
           [expertise gregs java 3]
           [expertise gregs politics 3]
           [expertise gregs thesis 1]
           [expertise gregs money 3]
           `[deadline gregs ,deadline-list6]
           [busy-inherent gregs 2]
           [cost-inherent gregs 2]
           [speed-inherent gregs 3]
           `[free-time gregs ,free-list6]])

(tell [and [location-of jcma ne43-807]
           [type-of jcma person]
           [age jcma 35]                           ;Again a guess...
           [expertise jcma lisp 5]
           [expertise jcma hacking 5]
           [expertise jcma thesis 3]
           [expertise jcma brainstorming 4]
           [expertise jcma rock-climbing 5]
           `[deadline jcma ,deadline-list7]
           [busy-inherent jcma 4]
           [cost-inherent jcma 4]
           [speed-inherent jcma 5]
           `[free-time jcma ,free-list7]])

(tell [and [location-of rhu ne43-807]
           [type-of rhu person]
           [age rhu 45]                            ;estimate
           [expertise rhu politics 5]
           [expertise rhu money 4]
           [expertise rhu infostructure 5]
           `[deadline rhu ,deadline-list8]
           [busy-inherent rhu 2]
           [cost-inherent rhu 2]
           [speed-inherent rhu 3]
           `[free-time rhu ,free-list8]])

(tell [and [location-of reti ne43-703]
           [type-of reti person]
           [age reti 40]                                   ;estimate
           [expertise reti lisp 5]
           [expertise reti hacking 5]
           [expertise reti MacIvory 5]
           [expertise reti VLM 5]
           `[deadline reti ,deadline-list9]
           [busy-inherent reti 4]
           [cost-inherent reti 4]
           [speed-inherent reti 5]
           `[free-time reti ,free-list9]])
```

W:>GSDOLINA>thg2>resource-knowledge.lisp.4

```
(tell [and [email-address gsdolina gsdolina@mit.edu]
           [email-address hes hes@ai.mit.edu]
           [email-address rladdaga rladdaga@ai.mit.edu]
           [email-address blumberg blumberg@ai.mit.edu]
           [email-address cvince cvince@mit.edu]
           [email-address gregs gregs@ai.mit.edu]
           [email-address jcma jcma@ai.mit.edu]
           [email-address rhu rhu@ai.mit.edu]
           [email-address reti reti@ai.mit.edu]])

(tell [and [type-of fax1 fax]
           [fax-number fax1 617-253-5060]
           [location-of fax1 ne43-830]])

(tell [and [type-of the-real-paper printer]
           [printer-properties the-real-paper 6 0 4]
           [location-of the-real-paper ne43-830]
           [can-print polk the-real-paper]
           [can-print polk phoenix]
           [type-of phoenix printer]
           [printer-properties phoenix 6 0 4]
           [location-of phoenix ne43-830]
           [printer-cost the-real-paper 3]
           [printer-cost phoenix 3]])

(tell [and [type-of polk computer]
           [location-of polk ne43-803]])

(tell [and [has-stuff gsdolina pager 1800-SKY-PAGE_pin_8704376]
           [has-stuff gsdolina cellular 617-462-1910]
           [has-stuff jcma pager 617-263-9041]])
```