Decision Theoretic Resource Management for Intelligent Environments

by

Robert W. Kochman, Jr.

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

© Robert W. Kochman, Jr., MMIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author

Department of Electrical Engineering and Computer Science
May 16, 2003

Certified by

Dr. Howard Shrobe
Principal Research Scientist
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Students

ARCHIVES
Decision Theoretic Resource Management for Intelligent Environments

by

Robert W. Kochman, Jr.

Submitted to the Department of Electrical Engineering and Computer Science on May 16, 2003, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Intelligent environments are becoming more complex, involving greater numbers of more sophisticated devices and agents. As complexity increases, it becomes difficult to build agents that effectively use the resources in their space. Agents may not know which resources are present in a given space, and they are almost certainly do not know which resources are inoperative or being used by other agents. To address this problem, this paper introduces PrefMan, a new resource management system for intelligent environments. To use this system, agents simply request a particular service type and give preferences over resource attributes. The system keeps track of resources, their types, and their availability, and uses the preferences to intelligently allocate resources to the requesting agent. This paper describes PrefMan in detail and illustrates why it is an effective solution to the resource management problem.

Thesis Supervisor: Dr. Howard Shrobe
Title: Principal Research Scientist
Acknowledgments

I’d like to thank...

...my thesis supervisor, Dr. Howie Shrobe, for countless ideas and direction on this project.

...my parents for their support, financial and otherwise.

...my friends for keeping me sane during my time here at MIT.
Contents

1 Introduction .................................................. 13
  1.1 The Problem .............................................. 14
    1.1.1 Resource Management ................................ 14
    1.1.2 What This Problem is Not ............................ 14
  1.2 The Solution ............................................. 15
  1.3 Scenarios ................................................ 15
    1.3.1 Showing a Movie ..................................... 15
    1.3.2 Sharing the News .................................... 16

2 The Problem ................................................ 19
  2.1 System Requirements .................................... 19
    2.1.1 Agents Request Services, Not Resources ............ 20
    2.1.2 Resources Allocated Based on Generic Preferences of Agents . 20
    2.1.3 System Arbitrates Among Agents ..................... 20
    2.1.4 System Can Operate in a Variety of Spaces .......... 20
    2.1.5 Most Requests Can Be Fulfilled In Several Ways ...... 20
    2.1.6 Constraints Among Resources Must Be Considered .... 21
    2.1.7 Resources Added and Removed Easily ................. 21
  2.2 Metaglue-specific Requirements ......................... 21

3 Design Overview .......................................... 23
  3.1 Resource Abstraction Level .............................. 23
  3.2 Resource Representation ................................ 24


7
3.2.1 Features ................................................. 24
3.2.2 Feature Sets ............................................ 24
3.2.3 Availability ............................................. 24

3.3 Preferences and Utility .................................... 25
  3.3.1 Preference Structure .................................. 25
  3.3.2 Ceteris Paribus Preferences .......................... 26
  3.3.3 Utility Function Generation ......................... 26
  3.3.4 Ordinal Utility ....................................... 27

3.4 Overall Structure ......................................... 27
  3.4.1 System Components ................................... 27
  3.4.2 Resource Request Process ............................ 28

4 Implementation ............................................. 31
  4.1 Resource Allocator ...................................... 31
  4.2 Preference Compiler ..................................... 32
    4.2.1 Directed Graph Representation ................... 33
    4.2.2 Breadth First Search Algorithm .................. 34
    4.2.3 Utility Lookup ..................................... 34
  4.3 Resource Store .......................................... 34

5 Evaluation .................................................. 37
  5.1 Fulfillment of Requirements ............................ 37
  5.2 Initial Deployment ...................................... 39
  5.3 Future Work ............................................ 40
    5.3.1 Integration with Semantic Networks ............... 40
    5.3.2 Building Preferences ............................ 41
    5.3.3 Weighted Preferences ............................. 41
    5.3.4 Reducing Complexity .............................. 42
    5.3.5 Sharable Resources ............................... 42
    5.3.6 Resource Scheduling .............................. 42
  5.4 Lessons Learned ........................................ 43
5.4.1 There Exists No Perfect Solution .................................. 43
5.4.2 Dangers of Extending an Existing System ..................... 43

6 Comparison With Related Systems ..................................... 45
  6.1 Intentional Naming System ......................................... 45
  6.2 Open Agent Architecture ............................................ 46
  6.3 Rascal ................................................................. 46

7 Contributions .................................................................. 49
  7.1 Preference-Based Allocation ......................................... 49
  7.2 Service Hierarchies ..................................................... 49
# List of Figures

1-1 Example of explicit resource type request. .......................... 18
1-2 Example of a more general service request. ............................ 18
3-1 Block diagram of the overall system structure. ....................... 28
4-1 Example of a Preference Set and the corresponding utility graph. ... 33
4-2 Resource Store data structure. ............................................. 35
5-1 How an agent requests a resource from another space. ............... 39
Chapter 1

Introduction

An intelligent environment (IE) is a space augmented with devices and logic that allow high-level interaction with users. In such spaces, users can typically perform simple tasks such as controlling the lights by speaking commands and asking questions concerning the status of a device. More sophisticated behavior is possible, such as anticipating a user’s needs and learning his preferences. Historically, traditional IEs have consisted of a single room, such as an office, conference room, or living space[2]. Spaces were always well-defined in terms of both scope and functionality.

Newer IEs are more sophisticated. Technology such as mobile devices and wireless networks muddles the boundaries of these spaces. One IE often seamlessly becomes another. The increasing ubiquity of these spaces raises new issues in the areas of coordination, privacy, and scalability. A resource management system today must adequately address these issues in order to be a workable solution[7].

This chapter provides a brief introduction to resource management and this system. Section 1.1 gives a general description of the resource management problem, specifically what it does and does not include. Next, Section 1.2 introduces this system as a solution to the resource management problem. Finally, Section 1.3 vividly describes two scenarios where the proposed resource management system greatly improves the operation of an intelligent environment.
1.1 The Problem

This section explores the need for a resource management system in intelligent environments. It specifies the resource management problem and also distinguishes this problem from similar ones.

1.1.1 Resource Management

In the past, it was possible to write applications for intelligent environments that would use resources directly. For example, a “show-movie” agent could be written for the particular DVD player, amplifier, and projector present in the room. Such an approach suffices for small prototype spaces, but as the scale and diversity of systems increase, so does the need for a more sophisticated system. Another problem arises when two applications need the same resources at the same time. If the “show-movie” and “show-presentation” applications are written to use the same projector, the two cannot operate concurrently, regardless of how many other projectors or other display devices are present. Similarly, if the necessary projector becomes unavailable (e.g. the bulb burns out), the applications that use it do, as well.

One predictable yet powerful improvement is to allow applications to use general resource types rather than specific resources. For example, a movie agent may use a generic projector interface rather than a particular projector, which makes the agent usable in any space that has a projector-type device. Adding yet another layer of indirection proves even more useful. Consider the same “show-movie” application. Suppose that it was written to use a “video display” rather than a projector. In this case, the application could use a projector or any other available display, such as a TV, monitor, or handheld device.

1.1.2 What This Problem is Not

It is important to note that this system is not intended to manage all things that may be considered a resource. For example, this system does not function as a traditional time-sharing system; it does not effectively manage processor cycles. It is also not
intended for use as a resource scheduler, though this is a related problem. See Section 5.3.6 for a discussion on how a resource management system might be extended to handle resource scheduling.

1.2 The Solution

This paper presents a solution to the resource management problem. The system, called PrefMan, is designed to fulfill the design criteria presented in Chapter 2. PrefMan is composed of three distinct parts:

- The **Resource Store** contains all information concerning resources, their attributes and current states.

- The **Preference Compiler** converts agent preferences to a utility function.

- The **Resource Allocator** uses the Resource Store and Preference Compiler to allocate resources to agents based on the agent needs and preferences and resource availability.

Each component is discussed in greater detail in Section 3.4.1 and Chapter 4.

1.3 Scenarios

This section describes two examples of effective resource management in an intelligent environment. These examples show the contrast between agent-resource interaction with and without the use of a resource manager. The second example shows how a hierarchical design extends the resource manager’s usefulness.

1.3.1 Showing a Movie

Suppose a user walks into a room and wants to watch a DVD movie. She places the DVD in the DVD player, and the audio system and projector turn on automatically. The image shows up on the main projector, but a window pops up saying that the
main room audio system is unavailable, so the audio track will be played on the user’s handheld device. The user acknowledges the notice, attaches her headphones to the handheld device, and begins to enjoy her movie.

Although the process is rather simple from the user’s perspective, there is much more going on in the background. When the user puts the DVD into the player, it starts the DVD agent. The DVD agent knows that it needs display and audio services in order to show a movie. The DVD agent submits a request to the resource manager for the two services and includes a set of preferences. The resource manager considers the available resources and the agent’s preferences and returns a set of resources, namely the main projector and the handheld device audio, to the DVD agent\(^1\). The DVD agent is designed such that using the handheld audio requires approval from the user. The user is willing to use handheld audio, so the DVD agent sends a message to the resource manager “claiming” the resources. The resource manager then marks those resources as owned by the DVD agent. When the movie is finished, the DVD agent releases the resources, and the resource manager once again marks them as available.

Although the process may seem complicated, it is important to note that the process would be even more complicated if the DVD agent had to obtain resources itself. It would have to have knowledge of the existence of all the resources it could use. It would also need the ability to use each individual resource’s interface, of which there could be many. The availability of each resource would also have to be ascertained. Is the best resource already in use? Was it removed from the space last month? The DVD agent itself would have to keep track of such changes. Clearly, the resource manager makes the DVD agent easier to implement.

### 1.3.2 Sharing the News

In this example, the user is sitting at his desk reading news stories at nytimes.com. He comes across an interesting story and wants to share it with his colleague down the

\(^1\)User preferences and agent preferences are essentially identical. The agent incorporates the user preferences, as applicable.
hall. He sends the story to that user's personal agent. Depending on what the user is doing, the personal agent could display the story on his computer screen, projector, or handheld device. It might also read the story over the speakers. It could scroll the text of the story on an LED sign. If the user is busy doing something else, it may simply send the story to his email or printer. How does the personal agent decide how to convey the information using the resource manager? There are two main ways.

1. Send the resource manager a request for a particular class of resources (e.g. projector or LED sign), as shown in Figure 1-1. This is the familiar case where the resource manager is given a request for a set of particular resource types (mapped from a service request) and the resource manager returns a set of resources that fulfill these types. Such an organization gives the personal agent greater control over how the user is notified.

2. Send the resource manager a "notify" request, including the URL of the story and a set of preferences. The idea here is that the preferences convey the status of the user. For example, if the user is on the phone, he would probably not want the story read over the speakers in the room. In this case, the resource manager not only decides which individual resources to use within a class, it decides which classes to use as well. See Figure 1-2 for a view of the arrangement. This arrangement abstracts away even more of the problem, allowing the personal agent to be quite simple.

Clearly, there are significant advantages to this multi-tiered arrangement.
Figure 1-1: Example of explicit resource type request. The agent requests a projector, and the resource manager assigns Projector A.

Figure 1-2: Example of a more general service request. In this case, the agent requests a notification. The resource manager, considering the agent’s preferences, decides that a projector is the best resource type for the job. It then looks at the available projectors and selects Projector A.
Chapter 2

The Problem

This chapter explores in detail the problem that the resource manager solves. Section 2.1 discusses the high-level requirements of a resource manager for an intelligent environment. Then, specific requirements derived from use of the Metaglue system are discussed in Section 2.2.

2.1 System Requirements

There are several major requirements that the system must fulfill. They are as follows:

- Agents request services, not resources.
- Resources are allocated based on generic preferences of agents.
- System arbitrates among agents.
- System can operate in a variety of spaces.
- Most requests can be fulfilled in several ways.
- Constraints among resources must be considered.
- Resources may be added and removed easily.

The following sections discuss each requirement in detail.
2.1.1 Agents Request Services, Not Resources

As discussed earlier, it is extremely advantageous to be able to build agents that do not rely on the resources and configuration of a particular space.

2.1.2 Resources Allocated Based on Generic Preferences of Agents

Agents need not concern themselves with the attributes of particular resources; rather, they express preferences for certain sets of attributes over others. These generic preferences are easy for agents to construct and require no knowledge of particular resources. For example, if an agent needs a fast computer, it does not need to know which computers are fast. Instead, it simply includes “speed fast” in its list of preferences.

2.1.3 System Arbitrates Among Agents

An agent need only concern itself with its own needs. For example, the movie agent need not worry that some other agent may have a greater need for a projector than it does. The resource manager alone makes these decisions based on the needs and preferences of the individual agents.

2.1.4 System Can Operate in a Variety of Spaces

The original Metaglue-enabled resource managers were designed mainly for use in the Intelligent Room. PrefMan, though still closely tied to Metaglue, should be equally useful in spaces of all sizes with resources of any imaginable type.

2.1.5 Most Requests Can Be Fulfilled In Several Ways

In addition to selecting the right resource types for a particular service, a large part of the resource manager’s purpose is to select the best resources within each type. If only one resource of each type is present, then there is not much choice; a sparse
resource environment calls for a simpler system. Therefore, we tailor this resource management system to environments where choosing the right one of many resources is critical.

2.1.6 Constraints Among Resources Must Be Considered

There are certain situations where a set of resources from appropriate resource types may not be feasible. For example, suppose an electronic slide projector service uses the following resource types: computer, video mux, and projector. Clearly, the system cannot simply select the most appropriate resource from each category, because there is the additional requirement that the computer and projector must be attached to the same mux. Additionally, the mux must not be in use. Situations like these are not uncommon, and as such, must be effectively handled by this system.

2.1.7 Resources Added and Removed Easily

The available resources in newer intelligent environments are constantly changing, largely due to the increasing number of mobile devices in use. A handheld PDA could be available for a few seconds, while someone’s laptop computer could be present for days. Also, electronics are notoriously prone to failure, which means that even permanently installed devices can instantly become unavailable without notice. Due to the dynamic nature of these spaces, this system must allow the addition and removal of resources in a simple, low-cost manner.

2.2 Metaglue-specific Requirements

Just as resource managers are adjusting to work in a variety of spaces, so is Metaglue. With each successive release, Metaglue is better able to operate in a variety of environments. As such, this system must be designed to work closely with Metaglue and take advantage of its methods of dealing with agents.
Chapter 3

Design Overview

Having explored the system requirements, we begin to see how they might be fulfilled. This chapter describes a high-level system design and illustrates how the design is a foundation for an effective solution to the resource management problem. Section 3.1 suggests how individual resources should be abstracted away from the user agents. Next, Section 3.2 discusses the resource representation. Preferences and utility, including utility function generation, are discussed in Section 3.3. Lastly, Section 3.4 shows the overall structure of the system and illustrates the steps of the resource request process.

3.1 Resource Abstraction Level

As discussed earlier, the main goal of the system is to free agents from concerning themselves with the potential problems associated with using resources, namely resource discovery and arbitration. There are many ways to design the abstraction, with varying degrees of detachment from the actual resources. As we already saw in Section 1.3.2, there are significant advantages to allowing multiple levels of abstraction. One simple way to do this (and the method used here) is to build a hierarchy of services, called a “service tree.”

In order to build a service tree, we must first understand that services are composed of resources and other services. For instance, a DVD-movie service uses DVD-
player, video, and audio services. The video service, in turn, uses a projector resource and a mux resource. Although this example spans only two levels, it is easy to see how such a tree could be an arbitrary number of levels deep.

3.2 Resource Representation

This section describes how each individual resource is represented in the system.

3.2.1 Features

A Feature (also often called an attribute or property) represents a piece of information about a resource. Each Feature consists of two parts, the name and the value. The name designates the type of characteristic that the Feature describes, and the value describes that resource’s embodiment of the characteristic. The pairs “speed fast” and “quality high” are simple Feature examples.

3.2.2 Feature Sets

A Feature Set is simply an unordered collection of Features. For the purposes of the resource manager, a resource’s Feature Set completely describes the characteristics of that resource. Therefore, every resource has exactly one Feature Set.

3.2.3 Availability

Naturally, a resource must be available in order to be assigned to an agent. Therefore, the system must have some way to represent the availability of a resource. For the sake of simplicity, this system only keeps track of whether a resource is available. Therefore, there are only two possible availability state transitions:

available $\rightarrow$ unavailable When a resource is assigned to an agent or an external entity (such as a diagnosis and recovery system) marks the resource as inoperative.
available → available When an agent is finished with a resource and returns it to the pool or an external entity marks the resource as working again (e.g. after having been repaired).

Such a simple representation allows the system to focus more on the resource allocation itself, leaving more complicated fault diagnosis and recovery issues to external entities. One interesting related issue deals with resources which can be shared to some extent. For example, a video multiplexor may have 10 inputs and two outputs, which means that two different, unrelated agents could share this resource. Section 5.3.5 explores how this system could be modified to deal with such resources.

3.3 Preferences and Utility

As explained earlier, agent preferences are critical in allocating resources intelligently. Effective use of preferences requires a robust preference representation. Likewise, the utility representation must also be sound. This section describes the structure of both preferences and utility used in the system.

3.3.1 Preference Structure

In order to do anything useful with preferences, we must define a structure to represent preferences. Because we are ultimately speaking of preferences for one set of resources over another, it is necessary that preferences and resources share some aspects of their representation.

Features

Features (and also Feature Sets) are the link between preferences and resources. For instance, a projector may have “high quality” and an agent might favor projectors with “high quality.” Such commonality is crucial in mapping resources to utility values.
Preferences

A Preference is a weighted relation between two Feature Sets. The weight describes "how much" one is preferred over the other. As discussed below, preference weights are a tenuous and arguably invalid idea. Possible valid uses for weights are discussed in Section 5.3.3.

3.3.2 Ceteris Paribus Preferences

When we talk about one feature set being preferred to another, we are speaking in terms of ceteris paribus, meaning "other things equal" [11]. This distinction is perhaps best explained by a simple example. Suppose a user prefers high speed to low speed and high quality to low quality. We might write this as

\[ \text{fast speed} \succeq \text{slow speed} \quad \text{and} \quad \text{high quality} \succeq \text{low quality}. \]

With these preferences, we know the following is true:

\[ \text{fast speed} \& \text{high quality} \succeq \text{slow speed} \& \text{low quality}. \]

Although the above case is clear, the following is more difficult:

\[ \text{fast speed} \& \text{low quality} \equiv \text{slow speed} \& \text{high quality}. \]

What do we know about the above relation from the initial preferences? Nothing. In order to make sense of this relation, we need more information in the form of preferences over these particular feature sets or an indication of whether speed or quality is more important. Using information concerning relative importance of features in building utility functions is discussed briefly in Section 5.3.3.

3.3.3 Utility Function Generation

The creation of a utility function from agent preferences requires ensuring that a Feature Set that is preferred over another Feature Set has a higher utility (U). Formally, for Feature Sets \( FS_1 \) and \( FS_2 \),

\[ U(FS_1) \geq U(FS_2) \iff FS_1 \succeq FS_2 \]  \hspace{1cm} (3.1)
where \( FS_1 \succeq FS_2 \) indicates that \( FS_1 \) is weakly preferred to \( FS_2 \).

It is important to note that this compilation of a set of preferences into a utility function occurs before a service request is made. In other words, the utility values are pre-computed, essentially creating a table of all possible feature sets and utility values. Therefore, finding the utility of a particular set of resources requires simply matching those resources' features to the feature set in the utility function table, and reading off the value. Such a design greatly improves the speed of fulfilling resource requests.

### 3.3.4 Ordinal Utility

It is important to note that the notion of utility is valid up to a monotonic transformation only. Specifically, this means that

\[
U(FS_1) \succeq U(FS_2) \iff f(U(FS_1)) \succeq f(U(FS_2))
\]

(3.2)

where \( FS_1 \) and \( FS_2 \) are feature sets and \( U \) is a monotonic function\(^1\). Intuitively, this means that utility values simply tell us which feature sets are preferred over others, but not "how much" one is preferred to another[12].

### 3.4 Overall Structure

This section gives a brief overview of the system design.

### 3.4.1 System Components

Considering the required functionality of the system, there is a natural division into the following three components:

- The **Resource Allocator** interacts with Metaglue and the requesting agents

---

\(^1\)A monotonic (or order-preserving) function is any function \( f(x) \) s.t. \( \frac{df}{dx} > 0 \ \forall x \)
Figure 3-1: Block diagram of the overall system structure. Note that the Resource Allocator is the only component that interacts outside the system. Also, the Preference Compiler and Resource Store do not interact directly.

- The **Preference Compiler** takes agent preferences and converts them to utility functions. Also, given an agent’s preferences and a set of resources, computes the utility.

- The **Resource Store** contains information about each resource: its type, attributes, and whether it is available.

Figure 3-1 shows how these components interact with external agents and each other.

### 3.4.2 Resource Request Process

The Resource Manager operation centers around agents’ request and use of resources. The following is an overview of how agents deal with resources and the resource manager:

- First, the requesting agent specifies the requested service and its preferences to the resource manager.

- The Resource manager then selects groups of resources based on the resource types required for the particular service.

- The agent preferences are then used to construct a utility function.
• Next, the resource manager determines best set of resources (namely, the one with the highest benefit/cost ratio) and returns it to the requesting agent.

• If the requesting agent approves, it claims the resources from the resource manager.

• When the agent no longer needs the resources, it notifies the resource manager, which returns them to the resource pool.
Chapter 4

Implementation

This section describes the implementation of the Resource Manager. The Resource Manager has three distinct components, the Resource Allocator, Preference Compiler, and Resource Store. The following sections discuss each of these in detail.

4.1 Resource Allocator

The Resource Allocator (RA) is the centerpiece of the system. All agents interact with this component, and it is the only component that interacts with the Preference Compiler and Resource Store. These two components handle the bulk of the functionality, so the resource allocator is rather simple. Nonetheless, it must intelligently use the other two for the system to operate effectively.

The Resource Allocator component is a subclass of the ResourceManager class built at the same time as Rascal[5, 6]. As such, it provides much the same functionality as Rascal. Naturally, the most important piece is that of the resource request and allocation component. The operation of this component is simple and a largely linear process as follows:

Get utility function object from preference compiler. This step simply sends the preferences to the preference compiler and gets a utility function in return. All the real work occurs in the preference compiler.
Get candidate resource sets from resource store. The resource store takes a list of resource types and returns a list of resource collections that satisfy the requested types. Note that the resource store knows nothing of the agent’s preferences, so it makes no distinction among the resource sets it returns. However, the resource store does pass along the cost of each set of resources.

Find the resource set with the best benefit/cost ratio. The utility for each resource set is calculated by sending the set to the utility function object. The returned utility is then divided by the cost to yield the benefit/cost ratio. The resource set with the highest ratio is kept as the “best” set.

Construct a ResourceBunch and return to the agent. The ResourceBunch is the class that a ManagedAgent uses to deal with sets of resources.

Assign resources if agent approves. When the agent receives the ResourceBunch, it will decide whether it actually wants those specific resources. If so, it “claims” the resources from the resource allocator. The ResourceBunch and the requesting agent’s ID are sent to the resource store, which marks each of those resources as being used by that particular agent.

Note that this process is nearly identical to the one described in Section 3.4.2 as part of the design specification.

When the agent no longer needs a resource, it simply tells the resource manager the name of the resource it no longer needs, and the resource allocator passes this along to the resource store. See Section 4.3 below for a description of how the resource store tracks resource availability.

4.2 Preference Compiler

The Preference Compiler (PC) takes a Preference Set and assigns a utility (U) to each Feature Set in the Preference Set such that a Feature Set that is proffered over another Feature Set has a higher utility, as described in Section 3.3.3.
(speed fast) (quality medium) > (speed slow) (quality high)
(slow fast) (size 15) > (speed slow) (size 20)
(slow slow) (quality high) > (speed fast) (size 15)
(slow fast) (size 15) > (private true)
(slow fast) (size 20) > (speed fast) (size 15)

Figure 4-1: Example of a Preference Set and the corresponding utility graph. Note that successor nodes are preferred over their predecessors.

To find utility values, we represent the Preference Set as a directed graph and use a variant of the breadth first search algorithm (BFS).

4.2.1 Directed Graph Representation

Converting a Preference Set to a directed graph is simple. Each edge in the graph represents a preference of one Feature Set over another[11]. Therefore, the Preference Set is the list of edges in the graph. The nodes are simply the Feature Sets that appear in one or more of the Preferences in the Preference Set. Therefore, to build the list of nodes, simply iterate through the Preferences in the Preference Set and add any new Feature Sets to the list of nodes. Figure 4-1 shows how this graph would look for a simple Preference Set.
4.2.2 Breadth First Search Algorithm

As shown in Section 3.3.4, the utility values are ordinal rather than cardinal, which means that the actual values are irrelevant as long as Equation 3.1 holds for every pair of values. Therefore, when calculating the utility values, we can start with any node.

The algorithm starts with all nodes marked as unvisited. An arbitrary unvisited node \( n \) is selected and given the value 0 (i.e. \( U(n) = 0 \)). Then, each of \( n \)'s successors without assigned values are assigned the value \( U(n) + 1 \), which is 1 in this case. Similarly, \( n \)'s predecessors without assigned values are assigned \( U(n) - 1 \), or -1. The node \( n \) is then marked as visited, while its predecessors and successors are marked as semi-visited. Now, the algorithm selects one of the semi-visited nodes as \( n \) and repeats the process. Once there are no more semi-visited nodes, the algorithm looks for unvisited nodes, which would indicate that they are in a different connected component (and thus unreachable from the previous start node). The algorithm then runs on the new start node as before. Once all nodes have been visited, the algorithm terminates with all nodes having an assigned utility value[4]. For the sake of simplicity, the values are then normalized such that the lowest value is 0; that is,

\[
v'_i = v_i - \min(v_0, v_1, ..., v_n)
\]  

(4.1)

where \( v'_i \) is the \( i \)th original value and \( v_i \) is the \( i \)th normalized value.

4.2.3 Utility Lookup

Once utility values have been calculated for all Feature Sets contained in a given Preference Set, the task of finding a utility value for a particular Feature Set is a simple table lookup, as described in Section 3.3.3.

4.3 Resource Store

The Resource Store (RS) component contains all information concerning the resources in the system. Specifically, for each resource, the RS contains the following:
Figure 4-2: Resource Store data structure. The desired type is sent to the Type Table, which returns the vector of names of resources of that type. Each name then sent to the Name Table, which returns the ResourceBlob objects corresponding to the desired resources.

- Name
- Properties (i.e. Features)
- Cost
- Whether the resource is currently assigned to an agent
- ID of the agent to which it is assigned (if any)

For each resource, the above information is contained in a ResourceBlob object. Each ResourceBlob object is stored in a hashtable, keyed by the resource name and thus called the “name table.” This implementation alone would require iterating through each resource when looking for resources of a particular type. Therefore, for each resource type, there is a vector of resources of that type stored in a hashtable (the “type table,” keyed by type. Figure 4-2 shows the relationship.

Given a set of required resource types, the main task to the Resource Store is to return sets of resources containing one resource for each resource type in the request. Specifically, the RS returns every resource set that contains the proper resources given the resource set. For example, consider a request for one projector, one computer, and one audio device. If the Resource Store currently has available three projectors, two computers, and two audio devices, then it will return all 12 possible combinations of resources. Clearly a larger system with many resources of a given type would result in
an extremely large number of resource sets being returned. This problem is discussed in Section 5.1.

Lastly, the Resource Store handles requests to “claim” a set of agents. Basically, after receiving all the resource sets, the Resource Allocator identifies one set as the one that it wants to use to fulfill the request. The RS then marks all the resources in that set as “owned” by the requesting agent.
Chapter 5

Evaluation

Having designed and implemented a system based on a general problem and specific requirements, it is important to explore the quality of the solution. This chapter serves to revisit some issues mentioned earlier as well as to introduce some other. Section 5.1 evaluates the system in terms of each of the requirements set forth in Chapter 2. Next, Section 5.3 explores ways in which the system could be improved. Lastly, Section 5.4 enumerates some of the pitfalls in designing and implementing such a system.

5.1 Fulfillment of Requirements

This section restates the system requirements set forth earlier, and for each one, briefly describes how this implementation fulfills these requirements.

Agents request services, not resources. The system certainly fulfills this requirement. The service lookup table completely abstracts away from the agents the selection of the individual resources. Agents need only know which service or services provide the required behavior.

Resources allocated based on generic preferences of agents. As described earlier, agent preferences translate directly to a utility function, which is used to decide which resources best fulfill an agent's needs. This, of course, relies on
preferences being specified intelligently. The preference attributes must correspond to attributes of the relevant resources, otherwise the preferences and corresponding utility function will be useless.

**System arbitrates among agents.** Clearly, this system, rather than the agents themselves, arbitrates among the resources. Note that there is no mechanism for agents to consider the needs of other agents when making requests. This, of course, greatly increases the ease with which agents can be constructed. Undoubtedly, more globally efficient allocations could result from agent collusion; however, we are willing to accept this loss in efficiency in order to have a more simple system.

**System can operate in a variety of spaces.** Due to the generalized representations of services, resources, and preferences, this system is useful in many different spaces. As spaces become larger, so does the complexity of the system. Fortunately, larger intelligent environments are likely to be divided into smaller spaces. For instance, a building might be divided into floors. Each floor, in turn, would be divided into individual rooms. Each small environment would have a resource management system controlling its resources. In an agent in one environment needs a resource in another environment, the agent makes a request to the resource manager. The resource manager then makes a request to the manager in the other environment (see Figure 5-1). To the second IEs resource manager, the first manager appears as simply another agent making a service request. This ensures that all resources are always under direct control of the resource manager for that space.

**Most requests can be fulfilled in several ways.** As we have shown, the system considers all possible satisfying assignments of resources. Therefore, it is ideal for spaces where many resources of each type are present. However, because all resources are considered, the number of potential resource sets can become quite large if there are many resources of each type. See Section 5.3.4 for a discussion concerning how this complexity could be reduced in the future.
Figure 5-1: How an agent requests a resource from another space. The agent makes the request to its resource manager (A), which in turn makes a request to the other space’s resource manager (B) for that service. Resource manager B then selects the resource or resources and sends them back to manager A, which then passes them back to the requesting agent.

**Constraints among resources must be considered.** Intelligently created services are the key to ensuring that only compatible sets of resources are considered. Due to the hierarchical nature of the system, a simple service can be composed of many sub-services that capture the resource constraints. For example, a "projector" service can be composed of several sets of services, each of which represents one or more acceptable sets of resources.

**Resources added and removed easily.** The system has simple methods for adding new resources and removing ones no longer functioning. Additionally, when a resource becomes temporarily unavailable, it can be marked as such without permanently removing it from the system.

### 5.2 Initial Deployment

The first real-world test of this system is as a resource manager for Room 835, a new intelligent space in the AIRE group of the MIT AI Lab. The goal of this environment is to use user location, in the sense of regions in which sets of activities occur, as
a means of interaction. That is, each region represents similar user activities, and context-aware applications can use the knowledge of the user’s location to trigger particular application events[8].

This space is a particularly interesting application for PrefMan. In particular, user preferences are likely to change depending on where the user is located. As a simple example, suppose the user is currently sitting at his desk and the email agent is displaying his email, quite appropriately, on the computer screen. Then, suppose the user gets up and moves to the sofa. The system running in the room would likely change his display preferences to something more visible from the sofa. The system may also deduce that the user may not be interested in reading his email while on the sofa and change the preferences accordingly.

5.3 Future Work

Although this resource management system solves a problem, it also raises some questions. This section explores some potential changes and additions that could improve the system.

5.3.1 Integration with Semantic Networks

Within the AIRE group, Stephen Peters is developing a semantic network for representing various types of knowledge in intelligent environments[13]. Semantic networks allow information to be added and removed quite easily. Additionally, making inferences from the network is also simple, provided there is an appropriate interface for selecting the appropriate information. Even more important is that extracting preferences from a semantic network enables the use of information that is already available. Clearly, extracting existing data is more advantageous than requiring the system to duplicate it.
5.3.2 Building Preferences

Because PrefMan relies so heavily on user preferences, it works only as well if the preferences are comprehensive. Unfortunately, user preferences for a wide range of activities can be extremely complicated. One cannot reasonably expect a user to manually define hundreds of preferences over even more feature sets. Therefore, a system that intelligently builds preferences would be a very important complement to this system. A logical first step would be a simple system that observes and builds preferences based on a user’s activity (i.e. the system learns what the user likes). Viappiani and Pu are currently exploring various methods for capturing these preferences[14].

5.3.3 Weighted Preferences

The current implementation treats preferences as binary (either present or not) and utility values as ordinal (see Section 3.3.4). However, in reality, preferences are not binary; a user may prefer one thing over another to any degree. Additionally, people may care more about one preference than another. There is certainly some value in capturing this information. Representing the preferences is not difficult; simple values can be used to quantify the degree and significance of each preference. However, constructing a useful utility function from this information proves more difficult. Even if we assume that degree and significance values are combinable, using these quantified preferences to build a utility function raises some difficult questions. First, how does transitivity affect utility values? For example, if the user prefers feature set $A$ to $B$ with degree 2, $B$ to $C$ with degree 3, and feature set $A$ has a utility value of 4, what are the utility values of $B$ and $C$? If addition is used, they are 6 and 9, respectively. With multiplication, they are 8 and 24, respectively. What exactly does it mean to say that one feature set is preferred “two more” or “two times” to another? In fact, basic economic utility theory is largely based on the fact that utility cannot be viewed on an absolute scale. However, there has been substantial research in the area of cardinal utility, with some degree of recent success[9].
5.3.4 Reducing Complexity

The current implementation of PrefMan calculates the cost/benefit ratio for every possible combination of resources that fulfill the requested resource types. Clearly, this could become significantly time-consuming for spaces with large numbers of resources of each type. Although this system is not designed for large spaces (because large spaces should be subdivided into smaller spaces), it would be beneficial to make this calculation more efficient. Replacing the current "calculate everything" method with an intelligent search for an optimal allocation would be an excellent first step.

5.3.5 Sharable Resources

As discussed in Section 3.2.3, certain resources may be finitely sharable; that is, they can be used by multiple (but not an unlimited) number of agents. In the future, PrefMan should be equipped to handle such resources. In the case where there are no constraints among the uses of the resource, sharable resources can be represented as multiple instances of identical resources. Similarly, the used/unused status variable could be modified to indicate how many unused "slots" are remaining. In the case where resources can be simultaneously used by several agents within certain constraints, the resource allocation framework would have to be modified to consider such constraints.

5.3.6 Resource Scheduling

Currently, PrefMan considers only current resource needs; that is, it does not try to anticipate an agent's future resource needs. Not surprisingly, the quality of allocations could be improved by considering this information. The Planlet project seeks to represent user tasks and associated plans, so by using Planlet to obtain this information, the resource manager could predict which resources would be needed at different times in the future and plan current allocations accordingly[10].

42
5.4 Lessons Learned

In the course of designing and implementing this resource management system, a few facts become apparent. This section discusses two of the most notable lessons learned throughout the development process.

5.4.1 There Exists No Perfect Solution

The design of a resource manager involves one tradeoff after another. For instance, the resource abstraction level is a tradeoff between ease of building agents and the amount of control over resource selection. Choosing the number of resources to consider when fulfilling a service request also involves a tradeoff—namely that between quality of the solution and time needed to find the solution. When considering these or any other tradeoffs, one must make the decision based on the most common cases.

Consider the resource abstraction level tradeoff. In order to set the level at an optimal place, we look at how agents select resources. For instance, does the presentation service need a display, projector, or high-resolution projector? Does the agent need to know the size and brightness of the projected image, or is the resolution and orientation sufficient? We try to leave that decision up to the agents themselves as much as possible, but base decisions on how resources are most often used when necessary.

5.4.2 Dangers of Extending an Existing System

As discussed earlier, PrefMan is built within and around an existing resource management framework for the Intelligent Room[5, 6]. Although this system operates much differently than previous resource management systems in the Intelligent Room, much of the existing structure can be used. For instance, the methods by which agents make resource requests remain unchanged, except for the need for preference information. This arrangement allows the new resource management system to be integrated into existing intelligent environments with very minimal changes to agents already present.

The danger lies in trying to reuse too much of the existing structure. For example,
the initial implementation of PrefMan used the existing \texttt{Resource} class to represent resources. Problems arose when trying to integrate the new resource attributes into the old class; it was inefficient and complicated. The solution was to use the \texttt{Resource} class for certain purposes (such as returning resources to agents), but to add the \texttt{ResourceBlob} object, as described in Section 4.3. As in this case, it is important to reuse as much as possible, but note that sometimes existing components must be reimplemented to better suit the new system.
Chapter 6

Comparison With Related Systems

As shown earlier, the need for a resource management system for intelligent environments is clear. As such, several systems with similar design goals already exist. Looking at some of the most relevant, we see their strengths and weaknesses and how they compare to PrefMan.

6.1 Intentional Naming System

The Intentional Naming System (INS) seeks to give resources names based on what the resource “intends” to do. This idea of a resource’s intent basically corresponds to properties and attributes of that resource. Naturally, this naming system is in sharp contrast to more traditional naming methods (e.g. the Domain Name System) that largely use the physical location as a basis for naming[1].

Although INS appears to be a promising framework for large-scale system of resources, it is less appropriate as a resource management system for an intelligent environment. The descriptive naming scheme allows the system to convey the same attribute or feature information as PrefMan; however, the distributed nature of the name-resolving system may yield similar ideas expressed in several different ways. For example, a series of projectors could have the attributes “quality high,” “quality max,” “quality 98.” Each of these features values may convey roughly the same idea, but the lack of uniformity causes problems when comparing features for the purpose
of finding the best resource for a given request.

The INS does solve certain important problems that arise with trying to do large-scale resource discovery in a distributed environment. As IEs become even larger and more complex and as resource use across multiple IEs becomes more prevalent, the sophisticated resource discovery tools incorporated will certainly become more relevant. As it stands now, however, the simple, service-based approach used by PrefMan is more appropriate.

6.2 Open Agent Architecture

SRI's Open Agent Architecture (OAA) is an agent-based system that facilitates collaboration among agents in providing services. In this system, agents are either a "client" or "facilitator." Client agents perform some type of task, while facilitators match service requests to descriptions of clients' capabilities. Facilitators can also serve in additional capacities, such as a distributed data store[3].

As Gajos points out in his 2000 thesis, the major weakness in OAA is its inability to divide tasks into subtasks[5]. Viewing tasks as resources, this means that one request can yield only one resource. Additionally, fulfillment of a request is viewed as an instantaneous occurrence; that is, there is no idea of a task (or resource allocation) being "held" for some period. Clearly, this is an important omission. For instance, a projector resource must be held for the entire time that a presentation service is being provided.

6.3 Rascal

Rascal is the most recent full-scale resource management system developed for the Intelligent Room. It is built upon sound design principles, many of which it shares with PrefMan. For instance, PrefMan implements the same ResourceManager interface and uses the ManagedAgent class to interact with Metaglue. In short, this means that though quite different internally, PrefMan provides much of the same functionality
as Rascal[6].

The most important differences between PrefMan and Rascal concern the way in which resources are allocated. Unlike Rascal, PrefMan does not consider how much an agent "needs" a particular service when calculating the utility. This choice was made reasoning about utility and level of need is best done externally by an entity better equipped to make sense of such information. The two also differ in that PrefMan does not consider request bunches; rather, it wraps up multiple requests into a single service request. In the rare case that an agent needs several services that are not encompassed by a single service, the agent can query the resource manager to see which resources it could get, without actually requesting the resources themselves. This functionality allows agents to see their options before making an "all-or-nothing" resource request. Lastly, and perhaps most significant, are the differences in how the two systems consider agent preferences. Rascal determines the goodness of a set of resources by considering the closeness of the match between service properties and resource properties. PrefMan, on the other hand, only considers properties of the service to ensure that the resources actually fulfill the requirements of the service. Beyond that, the appropriateness of the resources is determined by the closeness of the match between agent preferences and resource properties[5].
Chapter 7

Contributions

This paper has presented a complete resource management system for intelligent environments. That said, there are two major advancements implemented in this system and described in this paper, namely preference-based resource allocation and service hierarchies. First, Section 7.1 discusses the impact of allocating resources based on simple agent preferences. Then, Section 7.2 considers the significance of representing services hierarchically.

7.1 Preference-Based Allocation

The centerpiece of RobMan's functionality is its ability to allocate resources to an agent using its preferences. This new ability allows agents to receive the resources they prefer, without having to explicitly give a preference ordering over all possible resources. Furthermore, preferences are given over resource attributes, rather than the resources themselves, which further detaches agents from the particulars of the resources that they use.

7.2 Service Hierarchies

Another important advance is representing services as a hierarchy. In this way, agents can request a service at any level, depending on what it exactly needs. An agent that
needs a projector will request a projector service. If the agent needs a display, but not necessarily a projector, it requests a display service. The display service may return a projector, but then again, it may not, depending on the agent’s preferences and resource availability. As new device technology emerges, new services can be incorporated into existing services. In other words, an agent built to use a display will be able to use new display technology, as long as the new display service is a sub-service of the original display service.
Bibliography


