ACCESS: Access Controls for Cooperatively Enabled Smart Spaces

by

Buddhika Kottahachchi

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

February 2005

Author .....
Department of Electrical Engineering and Computer Science
January 28, 2005

Certified by .....
Robert Laddaga
Research Scientist, MIT CSAIL
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

ACCESS: Access Controls for Cooperatively Enabled Smart Spaces

by

Buddhika Kottahachchi

Submitted to the Department of Electrical Engineering and Computer Science
on January 28, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Traditionally, access control mechanisms have been static in nature, requiring explicit intervention in order to change their behavior. However, since security requirements can be expected to change frequently and rapidly in ubiquitous computing environments, access control mechanisms that can react to the context in which they are applied are desirable. With this in mind, we have created ACCESS (Access Controls for Cooperatively Enabled Smart Spaces); a framework for enabling dynamic, role-based and context-aware access control mechanisms in ubiquitous computing applications.

Thesis Supervisor: Robert Laddaga
Title: Research Scientist, MIT CSAIL

Acknowledgments

This project would not have been possible without the advice and guidance of Dr. Robert Laddaga, my thesis adviser, who was always available to share his wisdom in matters both academic and otherwise. Gratitude is also expressed to Dr. Howard Shrobe, who believed in the project, supported it, and always seemed to know where the light was and how to reach it. I would like to thank Max, my friend and mentor, for coaxing me back in to grad school, introducing me to the AIRE Research Group, and encouraging me to answer the *interesting questions*.

I also appreciate the efforts of Prof. Roy Campbell from the University of Illinois at Urbana-Champaign, and Prof. Armando Fox from Stanford University, who took time out of their busy schedules to help me understand their own parallel research efforts.

To my mother, a special word of thanks for always believing and supporting me in my quest for knowledge. Thank you, for encouraging me to continue with my studies. It has been a most rewarding experience.

A word of thanks to my friends without whom my life would be incredibly dull: Samitha, for showing me the importance of being organized and on top of things; Jennifer, for making me believe that anything is possible, as long as you pursue it with passion, purpose, and conviction; Joyce, for introducing me to the game of golf - a surprisingly great stress reliever; Greg, Jim, Vivek, Berta, Aurora, Leslie, and the rest at Burton One for providing a home away from home; and Jeremy, my roommate, who endured the same pains as I did in meeting the graduation target we had set ourselves.

I would also like to thank Stephen, Kevin, Gary, and the others in the AIRE Research Group for taking me in, sharing their knowledge, and helping me focus and make progress on my own research efforts, as well as Andy and Tyler, my office mates, who helped ensure a stimulating work environment.

Finally, I would like to extend gratitude to the MIT Oxygen Alliance for helping fund this work.

Contents

List of Figures

Chapter 1

Introduction

We live in a world where technological advances are being made at a rapid pace. Moore's law[29] still holds true and drives computation to be faster, cheaper, and smaller. Thus, as the cost of computing becomes negligible, a multitude of possibilities for new applications emerge. Among them is a noticeable trend toward incorporating computing power in our physical environments. Yet as computing power becomes a part of our physical environment, we expose ourselves to new forms of security risk that weren't previously applicable. Interestingly, this also provides us with opportunities to address both these new risks and previously identified ones. This project explores these security concerns and examines how they can be mitigated, particularly within the context of ubiquitous computing systems.

In this chapter we introduce our research agenda by providing both an overview of the historical context and the motivational forces that drove the work described in the other sections of this thesis.

1.1 Ubiquitous Computing

Mark Weiser, in his seminal work describing a vision for the future of computing, said "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."[48] Subsequent work done by Weiser and others in extending this thesis led to the cre-

ation of the field of ubiquitous computing that studies means by which computing resources can be embedded in our physical environment and controlled through natural human interactions.

The field of ubiquitous computing focuses on changing the manner in which we interact with computing systems. Computers today require users to possess a great deal of knowledge about their interface and spend time focusing primarily on interactions with them before even the simplest tasks can be accomplished. This results in a context switch occurring, leading to the user focusing more on how to accomplish a task rather than on the task itself. Some of us have grown accustomed to these restrictions and have learnt to work around them. In doing so we have found computers to be very useful tools. Yet, to the vast majority of people in this world computers are still an alien tool, and significantly less useful than they could be.

In order to truly reap the benefits of technological progress we have made, we need to build systems whose operation and interface are more intuitive to us. We desire computers to become invisible, or at least recede to the periphery of our senses. We want them to be able to understand and use such natural human interfaces as speech, vision, etc. when interacting with us. Instead of having islands of computing in the form of individual computers, we desire computing itself to be an omnipresent yet shared resource that is as natural as the air we breathe[30]. Components of this resource should be able to work in concert to both determine what users are trying to accomplish and assist them in their tasks as needed.

Ubiquitous computing explores these desires and attempts to bridge the gap between our present reality and a utopia of computing. In doing so the user will be freed to focus on the tasks at hand as opposed to how they can be accomplished on his or her computer. This notion has been so radical that it is now described as the “third great wave in computing” behind the mainframe and personal computing eras[49].

1.2 Smart Spaces

Smart spaces are a stepping stone toward the ultimate goal of ubiquitous computing. They are isolated environments with significant computing resources embedded within them. By working in concert with a range of sensors these computing resources have the ability to observe the physical world, interpret those observations, perform reasoning on these interpretations, and then perform actions based on the reasoning. When computing and sensor resources are put together in such a manner within a physical space we refer to them as "Smart Spaces". They are also referred to as intelligent environments, reactive environments, aware spaces, etc. in the literature.

1.3 Security and Privacy

When our environment becomes enabled in the manner described above, we open up opportunities for both our privacy and security to be compromised. Of particular concern is the scenario where computing resources embedded in our environment fall under the control of unscrupulous elements.

If these computing resources are able to track our location and the very actions we perform, then should anyone who so desires have access to that information? Arbitrary access raises immediate concern regarding individual privacy. So it becomes apparent that while we wish to share information, we care about who we share that information with. For example, Alice may wish to share her calendar with her co-worker Bob. However, Chris, who is unknown to both Alice and Bob, shouldn't have the same level of access that Bob has to Alice's calendar. Similarly, a conversation initiated and conducted over video conferencing or some other suitable resource available in one's environment should only be viewed or heard by the intended participants. But if the resources we use don't belong to us, and therefore are not under our control, then can we trust them to act in our best interests?

Therefore, it becomes apparent that even while the intentions of ubiquitous com-

puting are pure, the haphazard deployment of ubiquitous computing applications should be avoided. While a number of research projects[22, 36, 9] have focused on building the infrastructure required to deploy smart spaces, they have been focused primarily on building systems that serve as proofs of concept. Even though these systems are beginning to reach levels of sophistication such that they may be applicable in production settings, the area of security and privacy in these systems is one that has been minimally explored thus far.

1.4 Access Controls

Access Controls are often considered the more trivial aspect of security systems. Yet, within the realm of Smart Spaces they take on a level of complexity and importance that rivals such fundamental aspects of security as cryptographic techniques. In ubiquitous computing applications, they provide a very important safeguard in protecting user privacy. Of course, authentication, privacy, and non-repudiation are important factors as well. However, a data transmission where the identity of both sender and recipient can be validated (authentication), the channel is encrypted (privacy), and we're able to validate that the data the recipient received did actually come from the sender (non-repudiation) is not useful if we cannot validate that the recipient was authorized to receive that data.

While authentication, privacy, and non-repudiation draw sustained research interest leading to progress, access controls have traditionally lagged behind. We believe that correctly determining access controls will be an important factor in building secure intelligent environments. Therefore, we chose to focus our efforts in the area of access controls. In particular, we explored means by which to provide robust, dynamic, role-based and context-aware access control mechanisms within intelligent environments.

Chapter 2

Background & Related Work

No pervasive computing project that seeks broad usage can long ignore security. MIT CSAIL's AIRE Project is no exception. This chapter will first provide a summary of related ubiquitous computing projects being pursued elsewhere. We compare and contrast their approaches to security to our own. It will be clear that while we think our approach differs, it has been influenced by their efforts. Finally, we present a historical overview of MIT's Project Oxygen[30] and the AIRE Group's Metaglue/Hyperglue project to provide an indication of where our own work is situated in the larger context.

2.1 Related Work: Gaia

Researchers at the University of Illinois in Urbana-Champaign have been working on the Gaia Project[36] since 1996. As such it is one of the more mature research efforts in this domain. The project was motivated by a lack of middleware through which to connect network enabled computing resources in building Active Spaces[1]. One of the key characteristics of Gaia is that it is viewed as an extension of our notions on conventional operating systems. As such, Gaia has three main components: the Gaia Kernel, Gaia Application Framework, and the applications themselves. The

[1]Active Spaces is a term used interchangeably with Intelligent Environments, Aware Spaces, Reactive Environments and Smart Spaces in the literature.

Gaia Kernel provides a management and deployment system for the distributed objects within the system, in addition to a set of basic services available to and used by all Gaia applications. Gaia applications are built on top of the Gaia Application Framework which provides a set of building blocks supporting features like the mobility, adaptation, and dynamic binding of objects within Gaia.

Furthermore, Gaia provides five basic services: Event Manager Service, Presence Service, Context Service, Space Repository Service, and the Context File System. The Event Manager Service is implemented as a decoupled communication model based on suppliers, consumers, and channels which is responsible for appropriately distributing events within an active space. The Presence Service uses a beaconing mechanism to maintain soft-state about the entities (e.g. Application, Service, Device, and Person) present in the space. In the Context Service a set of abstractions and tools to encapsulate both sensory and other data in capturing notions of context is provided. At the aggregated level this is represented using first order logic and boolean algebra to allow reasoning over contextual data. Information about the software and hardware entities providing useful resources within the space is stored and made available through the Space Repository Service. The Context File System provides mechanisms for personal data availability in a space in which a user is present.

Together, these provide mechanisms for resources (which are modeled as software objects) to be discovered, managed, and used by other resources. Inter-object discovery and communications are handled through the CORBA[18] infrastructure.

It is interesting to note that Gaia was designed from a systems perspective of the world. That is, attention was first paid to the lower level Gaia OS or middleware before applications were considered. Therefore, it appears to enforce constraints on the type of applications that can be built. The people behind Gaia look at it as an operating system for spaces, and perhaps due to intentional or unintentional bias on what form those spaces would take, have built a powerful framework which has limited applications. This may be why they have been limited to building applications within well defined self-contained spaces like conference rooms, lecture

rooms, and offices, but are currently unable to support interactions across such spaces. However, being one of the oldest and longest active projects in this domain they have contributed much toward our understanding of applications of ubiquitous computing in intelligent environments.

2.1.1 Security in Gaia: Cerberus

As a result of being a mature platform, the researchers involved in Gaia have had the opportunity to work on the security aspects of Smart Spaces[39, 4, 1]. As a result, some significant contributions have been made in this area. The key ideas emerging from this work are encapsulated in their latest iteration of security: Cerberus [1]. For example, they propose a multi-level authentication scheme that has associated confidence values describing the trustworthiness of the authentication achieved. This authentication scheme also decouples device and protocol abstractions to allow ease of extensibility. In the area of Access Control, where our specific interests lie, they highlight the importance of considering context in making access control decisions. However, it is important to note that Cerberus uses access control lists defined centrally by a system administrator. These ACL's are hierarchical in nature, and while being easy to maintain, lack flexibility.

2.2 Related Work: iROS

The Intelligent Room Operation System or iROS[34] is a meta operating system built as part of the Intelligent Workspaces [22] project at Stanford University. It is important in that it has already been deployed in a few production environments. For example, it is currently deployed in a public space at a library in Stanford which students are allowed to use. Furthermore, aspects of their design have been replicated by other projects like the i-Land[42] project at Fraunhofer-IPSI in Germany. The scope of iROS is intentionally restricted to Intelligent Workspaces. That is, iROS is geared for physical spaces that support collaborative work amongst indi-

viduals. Examples of such spaces include conference rooms and classrooms. Therefore, other applications such as offices or even residential settings aren't considered. Also, they consciously choose to refrain from performing complex inferencing with regards to user intention, but rather allows users to explicitly change the environment based on their needs. They also allow users to provide their own mediation and conflict resolution when sharing resources.

The general architecture of the iROS platform consists of three main components: the Data Heap, iCrafter[33], and the Event Heap[21]. The Data Heap is a central repository that applications can use to post data they care about. Once placed in the Data Heap other applications can access the data and the Data Heap will translate and return the data in a manner useful to the subscribing application. For example, if the data in question is a set of power point slides and a viewer application running on a handheld requests the data, the Data Heap will translate the slides into a collection of single JPEG[17] images for display on the handheld. The iCrafter component is used for system service advertisement and invocation, and also for generating user interfaces for services based on the subscribers constraints (i.e. type of display interface available, etc.). Finally, the Event Heap provides a centralized mechanism for communication between different applications. Communication happens as events which are posted on the Event Heap by the applications. Other applications can choose to subscribe to these events. All communication within iROS occurs through this Event Heap, thus creating a single point of failure. However, the general design choices and initial scope limitations made have ensured that iROS is a very manageable system that has performed remarkably well in its intended domain.

2.2.1 Security in iROS: iSecurity

Security, as with the Gaia project, is a recent area of focus in the iROS project. Current work in this area is a part of the iSecurity[41] effort. Like iROS, iSecurity has strictly defined scope which limits it to being a security solution for the iROS

platform used within interactive workspaces. iSecurity attempts to address both the issue of authentication and access control. Like most authentication schemes available, iSecurity also relies on a centralized mechanism described as an iSign server. A user logging into a device must first authenticate himself with the iSign server by conventional challenge/response mechanisms like passwords. A successful response yields a token of trust in the form of an X.509[50] certificate which is used thereafter to provide authentication with the Event Heap when posting and subscribing to events. It also allows for events to be transmitted via a secure tunnel using SSL[35]. We believe that while this provides privacy on the wire, the fact that all communications occur via the centralized Event Heap leads to security vulnerabilities (in addition to the reliability issues mentioned above) in the event that the Event Heap is compromised.

We found it interesting that iROS implemented a distributed mechanism for policy management and enforcement. Each device/application running within the iROS platform and making use of iSecurity's infrastructure is responsible for maintaining its own policies and enforcing them. This is desirable since it allows for applications to evolve based on their own requirements and not be constrained by a central policy enforcement and management engine. As such this is a design property that we found appealing and worthy of emulation.

2.3 The AIRE Research Group

Project Oxygen is an inter-disciplinary research effort at MIT's Computer Science and Artificial Intelligence Laboratory aimed at re-thinking how computing systems function and moving toward the realization of Weiser's original vision. As part of this project, researchers at MIT and its industry partners (The Acer Group, Delta Electronics, Hewlett-Packard, NTT, Nokia and Phillips) explore the boundaries of human-computer interaction, artificial intelligence, and the underlying supporting technologies.

The AIRE (Agent-based Intelligent Reactive Environments) Research Group is a

key member of the Oxygen initiative and contributes toward its goals by investigating the domain of Intelligent Environments utilizing a distributed agent architecture.

2.3.1 Early efforts

Interest in Smart Spaces at MIT began prior to 1994 and led to the SodaBot[6] project which took a software agent-based approach[8, 7] to enabling smart spaces. While these efforts were focused on system implementation aspects, shortly thereafter an HCI focus also emerged [44]. Thus, a dual-pronged approach of working on both system infrastructure aspects and the human visible applications that could run on the infrastructure was taken. Subsequently these efforts converged into the Intelligent Room project[3], which continues as a project within the AIRE Research Group.

2.3.2 Metaglue

Metaglue[9] is a distributed agent framework developed within the AIRE research group to facilitate the construction of Smart Spaces. It replaced the SodaBot framework. With the research focus within the AIRE group being in Smart Spaces, Metaglue has seen significant use since taking its current form in 1999. It has been deployed in a range of spaces including conference rooms[19], offices[13], and common spaces[25]. Metaglue provides for agent discovery and co-ordination. The most important component in Metaglue is the Catalog, which maintains knowledge about all agents and their capabilities. Thus, it provides the means for discovery when another agent requires a particular kind of service or resource. Once the catalog identifies the appropriate agent, it returns a stub to it and the requesting agent can access the desired resource or services via the agent stub. The agent stub hides inter-agent communication that occurs through JINI[43], making it seem like the resources and services on the other agent are locally available. Metaglue, also has sophisticated fault tolerance mechanisms that allow it to recover gracefully

from agent failures. The original design was inspired by the desire to make programming for Intelligent Environments easier via an extension to the existing Java language[15]. However, since then it has evolved and is now viewed as a middleware platform supporting Intelligent Environments. Nevertheless, its roots, being grounded in programming languages, yield a design that lends itself to being very easy to use from a programmatic perspective. Perhaps this is a reason for the large number of applications currently deployed using Metaglue.

2.3.3 Hyperglue

While Metaglue has been used extensively within different sub-projects in the AIRE group dealing with Intelligent Environments, with every new iteration they stretch the limits of Metaglue's capabilities. As such the AIRE group is presently augmenting Metaglue to add key aspects of functionality that were found to be desirable. Most notable among them is the ability to allow interactions between multiple distinct intelligent spaces. Currently, the architecture only supports independent and isolated distinct spaces. Furthermore, better abstractions for representing real-world entities are being provided. That is, it will be possible to associate a collection of agents with a particular entity. This effort and the resulting framework is called Hyperglue[31].

Security is another area of focus in Hyperglue, and as such this project provides some of the preliminary security work done for Hyperglue. Our work is on controlling access to the resources governed by the real-world entities represented in Hyperglue. A more detailed description of the technical aspects of Hyperglue and the manner in which this project fits into the Hyperglue framework are discussed in Chapter 5 (Software Architecture & Implementation).

Chapter 3

Project Overview

The previous chapters have provided an overview of the ubiquitous computing domain and argued for stronger security mechanisms within it. As stated before, this project focuses on the access control elements that are relevant in this domain. In this chapter we provide a high level overview of the approach taken and the scope of this project.

3.1 ACCESS: Access Controls for Cooperatively Enabled Smart Spaces

This thesis describes the design and implementation of the ACCESS (Access Controls for Cooperatively Enabled Smart Spaces) system. ACCESS is a dynamic, role-based, context-aware access control mechanism for Smart Spaces. ACCESS is both dynamic and context-aware because the constraints deemed applicable and evaluated in determining access rights vary according to the state of the environment. Furthermore, the environment state influences the roles assigned to entities requesting access rights and these roles have varying degrees of permissions associated with them. While ACCESS has been designed with Hyperglue in mind, we believe the level of abstraction provided is sufficient to enable its incorporation into other ubiquitous computing platforms.

3.2 Conceptual Requirements

Before we delve into the details, it is important to understand the conceptual requirements for enabling access controls in smart spaces. Here at the MIT Computer Science and Artificial Intelligence Lab, we have built a number of Smart Spaces as part of our ongoing research in the area [20, 19, 13, 25]. In doing so, it has become apparent that we are concerned with four primary types of entities: *People, Places, Devices and Data*. In our Smart Spaces, these entities can have agent societies associated with them. Furthermore, agent-based interactions occur between these entities both within a type and across types (i.e. between People and People, People and Places, People and Data, etc.). These entities we are concerned with can also share resources within their control during these interactions. The interactions themselves can be processes acting on behalf of the entities and thus representing them. While in general we work with entities that represent people, our framework should allow other types of entities to be represented as well. Furthermore, we are now interested in controlling access to these resources. This leads to a complex set of relationships and constraints, for which we would like to build an evolvable conceptual representation. As such we are interested in defining a set of basic concepts on which an ontology specifying our conceptualization of entities and their relationships to one another can be built.

3.2.1 Roles

We assert that when such entities interact, they assume a role for the purposes of that interaction[11]. We also assert that these roles can be and are often relational. That is, the roles are defined in terms of the relationships between the entities concerned.

For example, from the perspective of a Person Entity, another Person Entity could fit any of a range of roles such as:

1. Student - Fellow Student, My Student

2. Professor - My Adviser, Colleague

3. Friend

4. Administrator

5. Unknown Person, etc.

Also, from the perspective of a Place Entity, a Person Entity could fit a role such as:

1. User

2. Presenter

3. Administrator

4. Unknown Person, etc.

And, to a Data Entity a Person Entity could be a:

1. Owner

2. Subscriber

3. User, etc.

For example, Bob could be a *Fellow Student* from Carol's perspective but *My Student* from Alice's Perspective. Similarly, Alice could be a *Professor* from Carol's perspective but *My Adviser* from Bob's perspective. Furthermore, room 323 - a Place Entity - could be *My Office* from Alice's perspective and an *Office* from Bob or Carol's perspective. Thus, it is apparent that different entities can take on different roles from the perspective of other entities (Figure 3-1).

It is also important to note that an entity has the ability to assume more than a single role defined from another entity's perspective. For example, we know that Bob and Carol are *Fellow Students*, but they could also be intramural softball *Team Members* as well. In such cases, when making an access control decision a particular role may be more pertinent than others under a given context.

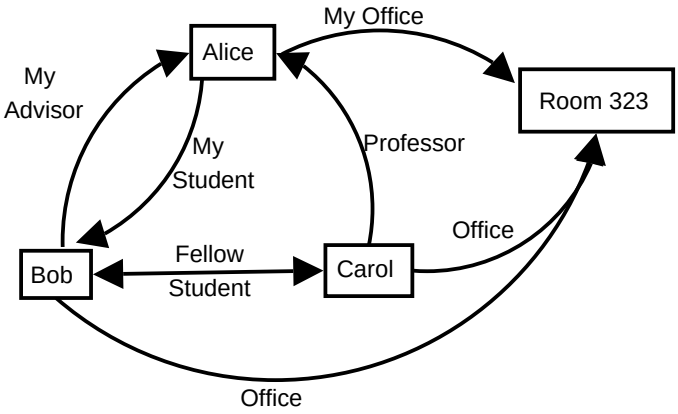

Figure 3-1: Role Relationships among different entities

3.2.2 Resources

These entities can also be in control of different resources. For example, a Place Entity such as a Conference Room can have Projectors, Lights, Sound Systems, Room Capacity Information as resources within its control. Similarly, a Data Entity like a Personal Information Management Server can have individual user Address books and Personal Calendars as resources within its control. A Device Entity such as a mobile phone can have such resources as a phone book, a speaker, a microphone and a display in its control. However, while these resources are within the control of particular entities, there can be instances when they are useful to other entities as well. For example, Alice could desire to check the Room Capacity Information of a conference room before she schedules a meeting for her Research Group. Once a suitable conference room is found, Alice would desire access to the Personal Calendars of Bob and her other students on the Personal Information Management Server in order to schedule the meeting.

3.2.3 Permissions

It is apparent that resources within the control of a given entity cannot be given out at will. For example, when Alice is scheduling her research group's meeting, should she be allowed to make an entry in Bob's calendar? What about Bob's right

to privacy?

Thus, we see the need for Permissions. Bob should be able to define that calendar access should only be given to those Person-Entities that assume the *My Adviser* role from his perspective. If Carol were able to do the same, then Alice, even though she is a *Professor*, would only have access to Bob's Calendar and not to Carol's. We would like to define permissions with respect to roles that entities can assume and the Resources they have access to. They could take the following form:

```
An entity assuming role X has access to Resource Y.
```

By access, we mean complete access. This may seem non-ideal. For example, one might desire to give *Write* access to one's Calendar for those assuming one role and *Read/Write* access to those assuming another role. However, cases belonging to this class can be covered by defining the resources at a more granular level: Calendar Write resource, Calendar Read/Write resource, etc.

3.2.4 Dynamic Access Controls and Context

The basic concepts defined thus far are sufficient to describe a set of predominantly static permissions. However, there is a need to consider dynamic changes to access permissions. Consider the case where Bob takes time off from school to deal with a family emergency. What he does during this time may be of a very personal nature to him, and even though Alice has static permissions to his calendar the current circumstances affecting Bob should be considered in making a decision on whether Alice should be allowed access or not. Thus, we have a need to dynamically model changes in the environment that affect access permissions.

Certain aspects of such context can be described by simply extending our notion of roles to include context. For example, we could define a role called *student-on-leave* and define permissions restricting access to the personal calendar of a person entity assuming that role. However, we believe that this approach may not be desirable from a scalability perspective. Instead, modeling context separately was a more attractive approach where we had a separate notion of context associated with each

entity. We assert that each entity should maintain a model of the context in which it currently finds itself. Thus, any decisions that require contextual information can be correctly determined.

3.3 Approach

Given this understanding of the conceptual requirements, we formulated our approach towards building ACCESS in the following manner. First, we selected resources as being the central element we would like to protect from arbitrary use. That is, a resource under the control of a given entity should be protected from arbitrary use by another entity.

Second, both roles and context were chosen to be the drivers behind access right evaluations. The roles in question here are those assumed by the resource requester. The context considered here is that in which the resource provider finds himself. While the context of a resource requester could be useful (ex. during emergency situations), a conscious design choice was made to exclude this factor from our model. The reason behind this is that we found no suitable means to guarantee the authenticity of the context cues provided by a resource requester. Especially in the context of Hyperglue, any claims the resource requester made about its context would have to be blindly trusted. Local context is only known by that entity and it allows rogue entities to misrepresent themselves to the external world. The only guarantees we can provide are with respect to the identity of the resource requester. Therefore, identity is a factor we consider in determining the roles that are applicable.

Third, we present an interrupt driven model (Fig. 3-2). That is, when a resource requester makes a request via ACCESS, ACCESS simply makes a determination of the requester's access rights at that time and either grants or denies the request. This can result in a resource requester who had previously been granted a resource having that permission revoked. Our design does not allow the polling of resource requesters to determine if they have completed using the resource in question. This

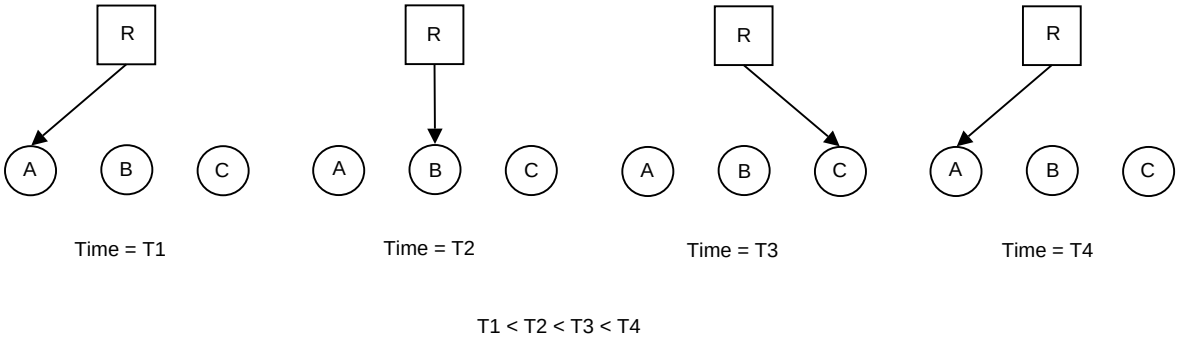

Figure 3-2: The resource requesters A,B and C can interrupt each other to obtain resource R. As shown here, at time T1 A has access to resource R, but at T2 B interrupts A to obtain it. C follows suit at T3 before A interrupts again and reclaims R at T4. This case holds true only if at each of the time instances, ACCESS was able to determine that the interrupting entities had access rights to resource R.

goes back to the point made above about the lack of trust that can be placed on requesting entities to represent themselves accurately. While this may seem non-ideal, we assert that it is possible to provide guarantees about a resource that is granted via the context infrastructure we use. For example, a resource provider can model a constraint that says a given resource can only be granted if it is currently not in use, or if it has been held by another entity for more than a stipulated time interval. At a more granular level you could even define the method call workflow that encapsulates a meaningful interaction with the resource and determine that its usefulness to the requesting entity has elapsed. Since all these items of knowledge can be maintained locally within the resource provider we are able to trust them and thereby circumvent some of the negative aspects of the interrupt driven model we propose. Another issue we need to address is the potential for deadlock in this model. What happens when two entities request the same resource at the same time? This issue can be addressed by implementing a two-phase locking mechanism inspired by those found in the domain of Database Systems.

In the next chapter we present the Knowledge Architecture used to both represent the access control policies we desire and also support inferencing on that representation. The following chapter provides the Software Architecture used in the actual implementation.

Chapter 4

Knowledge Architecture

Knowledge Representation plays a key role in ACCESS. In order to enforce the types of policies we described in Chapter 3 we desired a robust and extensible means for both representing policies and reasoning about them. In this chapter we describe the knowledge representation used in ACCESS and discuss the manner in which it meets our needs.

4.1 Access Control Policy Ontology

In order to structure our knowledge representation, we took the approach of defining an Access Control Policy Ontology. We felt an ontology was a suitable approach since it provides a set of guidelines that can be used to formulate a wide variety of policies. Furthermore, an ontology lends itself to creating a semantic network of the components on which inferencing techniques can be applied.

4.1.1 Design Goals

Our design goals were chosen to drive towards a simple, extensible, and useful knowledge representation for access control policies. We defined them as follows:

1. Make the ontology as simple and primitive as possible without causing undue burden on the user to create high-level constructs that would make it

useful. We understand that these are competing constraints, yet we believe a suitable compromise is needed. There exists a real danger of designing a representation that is too primitive for it to be useful to the average user or one that is too advanced such that it can only be manipulated in the manner it was designed for. The contrast between the assembly language - which can be used to build practically anything - with shell scripting languages - which have specific purposes - illustrates this.

2. Provide mechanisms by which extensibility can occur. In particular, we were concerned about being able to include all logic types that might ever be applicable in an instantiated policy. Instead, we would prefer to allow logic modules to be able to function as plug-ins to an instantiated policy, thus taking an embedded language approach.

3. Allow dynamic data to be factored in. We are working in Smart Spaces that are constantly evolving, and therefore need policies to be inherently dynamic. That is, we wanted to support the definition of dynamic policies that would function differently under different conditions.

In the next section, we look at some alternative policy definition approaches and compare them against our design goals to justify our own approach.

4.1.2 Alternative Approaches

A number of alternative policy definition mechanisms which could have been used in our efforts do exist. REI[23] is a policy definition language developed at the University of Maryland Baltimore Country which allows for policies to be described within OWL-Lite[46]. REI allows for the definition of permissions (both negative and positive) and obligations. It contains a rich set of semantic constructs to be used in defining policies. However, REI required its own engine for policy evaluation which wasn't easily compatible with the Hyperglue framework in which we were working.

XACML[14] is an XML-based policy definition framework. Its intended usage domain is in business applications. In our evaluation of XACML, we found it to be more suitable for defining static permissions. Due to the nature of static permissions, XACML provides excellent mechanisms for conflict resolution prior to run time. However, a policy evaluation engine suitable for our purposes was not available.

KAoS[45] is another effort at a policy definition language based on OWL[46]. Therefore, it leverages semantic web technologies, much like REI does, to provide an extensible means for policy definition.

Unfortunately, both REI and KAoS rely on the basic knowledge definitions being available and in their current forms don't support knowledge abstractions that can be made by an administrator of a specific domain in which the policies are applicable. Therefore, an assumption is made about the pre-defined existence of all knowledge in a friendly format that can be used in policy definitions. In reality, this isn't currently true. Furthermore, the use of this knowledge requires an inferencing engine that can reason over knowledge represented in OWL or other semantic web technologies.

Since such tools were under development, for the purposes of this project we decided to leverage tools that had been developed and used in-house. We will discuss our implementation in the next chapter, but in the rest of this chapter a description of our rudimentary knowledge architecture follows.

4.2 ACCESS Policy Ontology

The ACCESS policy definition ontology focuses on providing a framework for describing access control policies based on the relationships between resources, roles, and context. We assert that to sufficiently describe an access control policy governing a given resource, both the applicable role of the requesting entity and the applicable current context of the granting entity must be considered. Given this, the constructs we use in our policy definition ontology can be divided into three

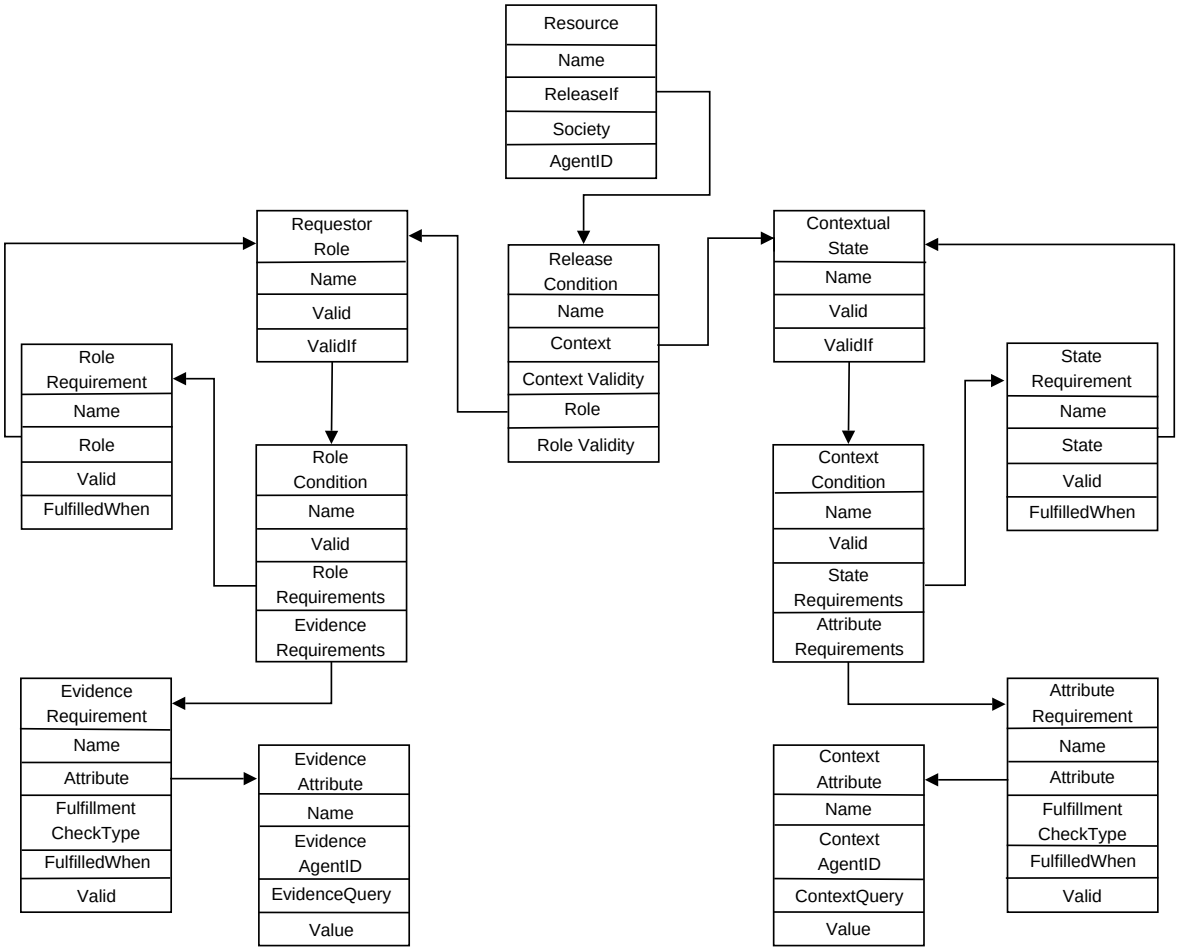

Figure 4-1: The ACCESS Policy Ontology

sections and are individually described below. A graphical representation of this ontology is presented in Figure 4-1.

4.2.1 Resources

Resources are elements - services, physical objects, virtual objects, etc. - of interest that are within the control of a particular entity. For example, my mobile phone is within my control. A light bulb in a room may be within the control of that room. Within the domain of smart spaces, these are the resources we would like to protect from arbitrary use.

Resource

A resource construct is used to describe a resource (usually represented by a software agent) of interest within a given agent society. This construct has the following properties:

- **Name** A name for the resource.
- **ReleaseIf** A list of ReleaseConditions. The fulfillment of one ReleaseCondition is sufficient to release the resource. This allows us to represent multiple situations under which a resource can be released.
- **Society** The agent society representing a real-world entity like a person, place, etc.
- **AgentID** The identifier for the particular agent that provides the software interface to the real-world resource we are protecting.

ReleaseCondition

This construct is used to provide a high-level representation of the constraints we need to consider when determining if a given resource should be released. It encapsulates a hierarchical tree structure of constraints that need to be satisfied. A ReleaseCondition has the following properties.

- **Name** A name for the release condition.
- **Context** If the ReleaseCondition takes provider context into consideration, the applicable ContextualState.
- **ContextValidity** A boolean indicating if the ContextualState referred to by the Context property is actually valid for the given case.
- **Role** If the ReleaseCondition takes requester role into consideration, the applicable RequesterRole.

- **Role Validity** A boolean indicating if the RequesterRole referred to in the Role property is actually valid for the given case.

4.2.2 Roles

Roles play an important part in determining access rights. If a requesting entity can assert and prove to a providing entity that he can assume a particular role the providing entity knows about, then the requesting entity is allowed the privileges associated with that role. Traditionally these have been static and worked on the premise that an entity proving its identity through proper credentials would be allowed to assume a given Role. For our purposes, this was determined to be too restrictive. Therefore, we use dynamic roles which can be built up using both other roles and evidence attributes. Evidence attributes are like the identity credentials mentioned above in that they support the requesting entity assuming a particular role. However, they are different in that they can be dynamic and can take into account other factors in addition to simple identity. For example, we might want to define a role of *room occupant*. This role would only be applicable to an entity that happened to be in the room in question.

RequesterRole

We use this construct to represent roles we would like to assign to requesting entities. These roles are defined from the perspective of the provider entity and therefore the requester has no knowledge of the properties of such a role. For example, Alice could define a RequesterRole of *colleague* and assign the right to her public calendar to it. If Bob can assume the role of *colleague* from Alice's perspective, he may be allowed to add an entry into Alice's Calendar. RequesterRoles have the following properties.

- **Name** A name for the requester role.
- **Valid** A boolean indicating if this Role is actually valid for a given case.

- **ValidIf** A set of RoleConditions that need to be satisfied for this Role to be valid in a given case. Note that *all* the RoleConditions listed here need to be valid for the Role to be valid.

RoleCondition

The RoleCondition construct is used to represent specific conditions that need to hold true for a particular Role to be valid. There are usually a set of RoleRequirements and EvidenceRequirements that contribute towards the validity of a RoleCondition. RoleConditions have the following properties.

- **Name** A name for the role condition.
- **Valid** A boolean indicating if this Role Condition is valid for a given case.
- **RoleRequirments** A set of RoleRequirements that need to be valid for a Role Condition to be satisfied for a given case.
- **EvidenceRequirements** A set of EvidenceRequirements that need to be valid for a Role Condition to be satisfied for a given case.

RoleRequirement

A RoleRequirement is used to represent whether a given RequesterRole needs to be valid or invalid for another RequesterRole to be valid. This allows us to define RequesterRoles in terms of other RequesterRoles in hierarchical structures thus providing for ease of use and eliminating redundant information. The following properties are contained within a RoleRequirement.

- **Name** A name for the role requirement.
- **Role** A RequesterRole we take into consideration when evaluating a RoleRequirement.

- **FulfilledWhen** A boolean indicating if the RequesterRole in the Role property needs to be valid for the RoleCondition to be fulfilled. This allows us to define RoleRequirements with negative relationships to the RequesterRoles.
- **Valid** A boolean indicating the validity of this RoleRequirement in a given case.

EvidenceRequirement

An EvidenceRequirement captures the affect an EvidenceAttribute might have on the applicability of a Role. For example, an identity credential can be an EvidenceAttribute that supports the notion that the entity presenting it can assume the *identity* role. An EvidenceRequirement contains the following properties.

- **Name** A name for the evidence requirement.
- **Attribute** An EvidenceAttribute whose value is important in determining the satisfaction of the EvidenceRequirement in a given case.
- **FulfillmentCheckType** An operator that evaluates the Attribute property in conjunction with the FullfilledWhen property to determine if the EvidenceRequirement is satisfied. Operators are currently defined to be checks for equality, greater than (or equal to), less than (or equal to), and inequality. In practice this has proven to be sufficient so far. However, the implementation does allow for extensibility with additional operators.
- **FulfilledWhen** The constraint used to evaluate the satisfaction of an EvidenceRequirement. Can mean different things depending on the value for FulfillmentCheckType as described above.
- **Valid** A boolean indicating the satisfaction of a EvidenceRequirement in a given case.

EvidenceAttribute

An EvidenceAttribute is an element of knowledge that may support an entity assuming a particular role. An EvidenceAttribute value may vary over time and is usually known by a software agent within a given agent society. We use this construct to store the means by which the current value of a given Evidence Attribute can be queried.

- **Name** A name for the evidence attribute.
- **EvidenceAgentID** The identifier for a software agent that has the means to acquire knowledge about this Evidence Attribute.
- **EvidenceQuery** The formulation of the query that is passed in to the agent pointed to by the EvidenceAgentID property such that the agent can correctly return the desired attribute. For example, we might query an Authentication-Agent with a 'CredentialCheck' query.
- **Value** The current value of this evidence attribute.

4.2.3 Context

Context is another key factor that ACCESS uses in making decisions on access rights. This is a recent development in access control mechanisms[10, 26]. Our constructs for representing context drivers are analogous to those used for representing requester roles. We argue that while an entity may be able to assume a role that has permissions to a given resource, the entity providing that resource may not be in a state that allows the release of such a resource. For instance, extending on the example above, even when Bob can assume the role of *colleague* if Alice is away on vacation and has closed her Calendar from public access, Bob maybe denied access to it. In order to handle this scenario we need to model the context drivers that affect Alice. One might argue that this can be folded into the Role definitions shown above. That is, we might define one role for *colleague* that is applicable when Alice

is on vacation and one when she is not. However, we believe that this leads to convoluted representation that when modeling complex scenarios can be difficult to use. Therefore, we model these two notions as separate factors.

ContexualState

We use this construct to represent contextual states applicable to the resource provider that are pertinent to access control decisions. These states are also defined from the perspective of the provider entity and therefore the requester has no knowledge of them. The ContextualState models the different states of a provider entity, thereby allowing it to be part of the consideration in making an access rights evaluation. The following properties exist within this construct.

- **Name** A name for the contextual state.
- **Valid** A boolean indicating if this ContextualState is actually valid for a given case.
- **ValidIf** A set of ContextConditions that need to be satisfied for this ContextualState to be valid in a given case. Note that *all* the ContextConditions listed here need to be valid for the ContextualState to be valid.

ContextCondition

A context condition captures the StateRequirements and the AttributeRequirements that need to be satisfied for a particular ContextualState to be applicable. The following properties are used to capture there requirements.

- **Name** A name for the context condition.
- **Valid** A boolean indicating if this ContextCondition is valid for a given case.
- **StateRequirments** A set of StateRequirements that need to be valid for a ContextCondition to be satisfied in a given case.

- **AttributeRequirements** A set of AttributeRequirements that need to be valid for a ContextCondition to be satisfied in a given case.

StateRequirement

Like a RoleRequirement, a StateRequirement is used to represent whether a given ContextualState needs to be valid or invalid for another ContextualState to be valid. This means that ContextualStates can be defined in terms of other ContextualStates in hierarchical structures which are easy to use and minimize knowledge redundancy. The following properties are contained within a StateRequirement.

- **Name** A name for the state requirement.
- **State** A ContextualState we take into consideration when evaluating a RoleRequirement.
- **FulfilledWhen** A boolean indicating if the ContextualState in the State property needs to be valid for the ContextCondition to be fulfilled. This allows the definition of StateRequirements with negative relationships with the ContextualStates.
- **Valid** A boolean indicating the validity of this StateRequirement in a given case.

AttributeRequirement

Similarly, an AttributeRequirement is used to represent individual ContextAttributes that may have a bearing on the applicability of a particular ContextualState. Like the EvidenceRequirement described above, an AttributeRequirement captures both the ContextAttribute and the logic that needs to be evaluated on the ContextAttribute to determine its influence on the applicability of a ContextCondition. They contain the following properties.

- **Name** A name for the attribute requirement.

- **Attribute** A ContextAttribute whose value is important in determining the satisfaction of the AttributeRequirement in a given case.
- **FulfillmentCheckType** The same functional and semantic behavior as that of the FulfillmentCheckType Property in the EvidenceRequirement.
- **FulfilledWhen** The constraint used to evaluate the satisfaction of an AttributeRequirement. Can mean different things depending on the value for FulfillmentCheckType as described earlier in the EvidenceRequirement.
- **Valid** A boolean indicating the satisfaction of a AttributeRequirement in a given case.

ContextAttribute

A ContextAttribute is an element of knowledge existing within the agent society that indicates that a particular ContextualState is applicable in a given case. Like an EvidenceAttribute, the value of a ContextAttribute may vary over time and is usually known by a software agent within the provider entity's agent society. We use this construct to store the means by which the current value of a given ContextAttribute can be queried.

- **Name** A name for the context attribute.
- **ContextAgentID** The identifier for a software agent that has the means to acquire knowledge about this ContextAttribute.
- **ContextQuery** The formulation of the query that is passed in to the agent pointed to by the ContextAgentID property such that the agent can correctly return the desired attribute. For example, we might query a ClockAgent with a 'DayOfWeek' query.
- **Value** The current value of this evidence attribute.

4.3 Analysis

We believe the ACCESS Policy Ontology described above meets our stated design goals. It provides the basic elements that are sufficient to define access control policies that consider both roles and context. The nature of these roles and context can be defined using this ontology and specific attributes that are available in the provider entity's agent society at large. Therefore, it can be expressive to the extent of the expressiveness of the environment in which it is deployed. However, it includes no higher level semantic constructs that make it easier to use in a particular domain and therefore restrict the manner in which the policy is defined. For example, one cannot claim that this ontology is only sufficient to describe Access Controls for a Database. Any application that desires to define access control policy based on requester roles and provider context can use this framework. It does, by design, have a bias towards applications within the Hyperglue framework. This bias is achieved by adding Hyperglue specific properties(ex. AgentID's, context queries) to the ontology. These properties can easily be modified and extended for use in other frameworks.

Furthermore, we support dynamic data by allowing attribute values to be obtained by querying the agents that may have knowledge about them at the time the values are required. This means our decisions won't be based on stale data but rather on current and relevant data.

It should also be apparent that the hierarchical structure of our ontology is inherently compatible with inferencing based on backward-chaining[37]. We believe that this is a mechanism for reasoning that is familiar to humans, and as such should make the task of defining or specifying policies more intuitive to the end user. It also allows for an efficient inferencing engine that makes evaluations based only on need, in contrast to one that lent itself towards a forward-chaining approach where wasted computation can occur.

In the next chapter we describe how this knowledge architecture was implemented in software and also how inferencing was performed on it to evaluate access

control policy decisions, or in essence how ACCESS was implemented in practice.

Chapter 5

Software Architecture & Implementation

In this chapter we focus on describing the software architecture used for the implementation of ACCESS. We begin by explaining how the ACCESS Policy Ontology described in Chapter 4 was represented and follow with a description of the mechanisms used for policy definition and persistence. Next, we discuss the inferencing mechanism built to evaluate policies and determine permissions. Finally, we describe how ACCESS was integrated into the existing Hyperglue framework to provide an access control capability and show an example of its use.

5.1 ACCESS Policy Ontology Representation

The knowledge architecture described in the preceding chapter required a software based representation to enable policy evaluations to occur. That knowledge architecture involves representing policies as interconnected ontology elements (or nodes). As such, it contains properties closely associated with a semantic network[38]. Therefore, we based our approach for the software architecture on the SEMANTIC [32] toolkit which had already been built and seen significant use within the AIRE research group.

5.1.1 SEMANTIC

SEMANTIC is a toolkit developed for the purpose of representing and working with semantic networks. It provides persistence through a relational database[5] at the backend. The relational representation uses sets of triples (similar to the RDF [47] triple) composed of: *unique identifier, property and value*. These triples represent resources or objects. It also allows for the use of these triples to instantiate Java objects with corresponding properties. Thus, the relational representation used for persistence can be instantiated as a collection of Java objects which provide ease of manipulation and navigation.

5.1.2 ACCESS in SEMANTIC

To allow ACCESS to leverage the SEMANTIC toolkit, we implemented Java classes for each of the ontology/knowledge elements defined in the ACCESS Policy Ontology. Each class contained the properties corresponding to the actual ontology element it represented. They were also augmented with components from the SEMANTIC toolkit to allow interaction with the persistent data storage mechanism. Furthermore, since Java allows object properties to point to instances of other Java objects, and since SEMANTIC is capable of serializing and storing such relationships, we had a means to represent semantic networks described using the ACCESS Policy Ontology as interconnected Java objects.

5.2 Policy Definition and Persistence

Having the ability to interact with an ACCESS policy as a semantic network built using interconnected Java objects was only useful if we could instantiate such networks as needed at runtime from a persistent storage mechanism. Thus, we needed a means to populate the persistent storage mechanism with ACCESS policies. We also needed an abstraction that allowed end-users to define ACCESS policies without being burdened by knowledge of the underlying implementation. To address

both of these needs we created PolicyGen.

5.2.1 PolicyGen

PolicyGen is a tool that:

1. Enables the definition of ACCESS policies using a set of primitive constructs
2. Generates an executable Java program that interacts with SEMANTIC to populate the persistent storage with the semantic network describing a given ACCESS policy.

PolicyGen is based on OntoGen[24]; a tool for instantiating and manipulating semantic networks in SEMANTIC. In creating PolicyGen, we identified a useful subset of the functionality and syntax available in OntoGen, and augmented it to meet our particular needs. For example, while OntoGen supports links between nodes in a semantic network as separate constructs, such links were represented using properties in the ACCESS policy ontology. Therefore, there was no need for links. However, since OntoGen didn't inherently support collections as properties of nodes and the ACCESS Policy Ontology relied on them, we included support for collections. The completed version of PolicyGen supports the syntax described below.

PolicyGen Syntax

- `is-a <instance_name> <node_type>`

  Creates an instance of an ACCESS policy ontology node (ex. Resource, ReleaseCondition, ContextualState, etc.). All node definitions must occur at the beginning of the policy description before any properties are defined. The PolicyGen parser enforces this to prevent overlap in the namespace.

- `property-value <instance_name> <property_name> <value>`

  Populates a specific property in a node described by the ACCESS policy ontology. The *value* given must be of the same type as that expected by the

property. Type definitions are made in the Java classes describing a particular node. The value can be a reference to an instance of another node defined within a given policy as long as the type requirements are met.

- `property-value-list <instance_name> <property_name> <value list>`

  Populates a specific property in a node described by the ACCESS policy ontology which is accepting a collection. The *value list* contains a comma separated list of references to instances of nodes defined in a particular policy.

In our use of PolicyGen we have found this to be a sufficient set of constructs to describe an ACCESS policy given knowledge of the expected ontology conventions. For an example, policy definition using the ACCESS policy ontology and for consumption by PolicyGen, see Appendix A. The example we provide is one that we have used in practice to protect resources within a Hyperglue agent society.

Using PolicyGen

The first step in using PolicyGen is to use the constructs described above to define an ACCESS policy like the one shown in Appendix A. Essentially, this amounts to describing a semantic network that conforms to the ontology described in Chapter 4 and which describes the types of constraints we would like to place on a set of resources. Once a policy has been created and stored in a *policy file*, we execute PolicyGen. If the *policy file* is well-formed, PolicyGen will create a new Java file called BuildSphere.java which contains code that:

1. Instantiates Java objects representing instances of particular nodes in the semantic network representing the ACCESS policy defined in the *policy file*.

2. Stores the instantiated Java objects to persistent storage via facilities afforded by the SEMANTIC toolkit.

Executing this generated BuildSphere.java file sets up an ACCESS policy for use within a Hyperglue society. The semantic network in persistent storage can be

instantiated in part or whole as Java objects through the SEMANTIC toolkit. We exploit this in the inference mechanism we built.

5.3 Inference Mechanism

Once we had a means to instantiate a policy representation, we required a mechanism to evaluate it to determine access rights at runtime. We exploited the semantic network's inherent support for backward-chaining and enabled backward-chaining by embedding intelligence in the nodes contained within the network. Our approach to enabling inferencing is described below.

5.3.1 Backward Chaining

As mentioned previously, the ACCESS policy ontology is conducive to backward-chaining. This is because at each node, the relevance of the node to an access rights evaluation is dependent on a set of other nodes (except for the leaf nodes like EvidenceAttribute, ContextAttribute, etc. in the ontology). Thus, given that we're determining access rights for a particular *Resource*, finding the node describing the permissions associated with that *Resource* needs to be done first. Thereafter, we need to evaluate the ReleaseConditions associated with it to determine if any of them hold true and access can be granted. The requirements for each of those conditions to hold true are described by other nodes like RequesterRoles and ContextualStates which must also be evaluated for validity. Thus, we chain backwards from the *Resource* node evaluating other nodes until we can determine that a particular ReleaseCondition holds true or that none of them do. Using this mechanism, an access rights evaluation can occur.

5.3.2 Embedded Intelligence

In performing the backward-chaining based inferencing described above, it is apparent that a mechanism for evaluating the relevance of a particular node must also

exist. We achieve this by embedding intelligence in each node. That is, we enable policy permissions to be self-evaluating. This is done by implementing an appropriate `evaluate` method in the Java objects that represent each node. The `evaluate` method is unique to each class of nodes and contains knowledge about the nature of the dependencies with other nodes and the state of that particular node required to ascertain its relevance in the inferencing mechanism. For example, a *Resource* node must find a ReleaseCondition that holds true in order to ascertain that the *Resource* can be released. The `evaluate` method on the *Resource* node sequentially obtains handles to the ReleaseConditions associated with it. If the ReleaseCondition has not been instantiated into memory it is retrieved from the persistent storage and instantiated. When a handle is obtained the `evaluate` method on that ReleaseCondition is called which in turn causes other nodes to be evaluated. This process is repeated till either a valid ReleaseCondition is found or all ReleaseConditions have been exhausted.

This is an artifact of the object-oriented nature of SEMANTIC and the influence it had on our own ACCESS ontology representation. By taking an object-oriented approach, our representation has an unambiguous meaning because the semantics are encoded in the data structure holding the knowledge. In doing so we have reduced the need for a co-ordinated inferencing mechanism (e.g. a rule chainer), instead allowing the knowledge representation to have the capability to determine the relevance of that knowledge when applied to a particular instance. That is, we're able to navigate to any node in the semantic network that represents knowledge about a particular ACCESS policy and query it for its validity or relevance. The required computation occurs in a distributed manner spreading from node to node as needed. This is in contrast to representations like the Semantic Web[2] which relies on external reasoners which traverse and interpret RDF graphs in their own ways.

5.4 ACCESS in Hyperglue

The work described in this thesis was positioned within the Hyperglue platform for enabling smart spaces. As such, in this section we will provide an overview of Hyperglue, it's security model and the manner in which we integrated ACCESS into Hyperglue.

5.4.1 Hyperglue

Hyperglue is a distributed agent infrastructure that provides lookup and brokering services to agents. These agents are organized into societies and Hyperglue facilitates inter-society communications. Furthermore, the notion of societies provide an excellent mapping for the real world (Place, Person, etc.) entities in which we are interested. Also, the agents provide an ideal mapping for the real world resources (Calendar, Mobile Phone, etc.) that interest us.

Figure 5-1 provides a description of how an agent request is made and fulfilled in Hyperglue. Initially *Agent Society 1* makes a request for an agent from its own Resource Manager. Each society contains a Resource Manager that is responsible for knowledge about the resources contained in the society and their availability/usage. Since resources within a society are held by that society, requests for resources within a society do not require any access control mechanisms. In the example shown in figure 5-1, the required resource is not available within the society. Therefore, the Resource Manager contacts its Ambassador agent. Each society has an Ambassador agent that represents the society to other agent societies and acts as an intermediary for brokering resources amongst them. Thus, the Ambassador agent representing *Agent Society 1* uses the Hyperglue Entity Directory to discover the required resource. The Hyperglue Entity Directory provides a distributed mechanism for resource discovery in any Hyperglue enabled agent society. Once the location of the required resource is established, the requesting society's Ambassador agent contacts the resource provider's Ambassador agent to obtain the resource. The provider society's Ambassador agent in turn passes that request onto

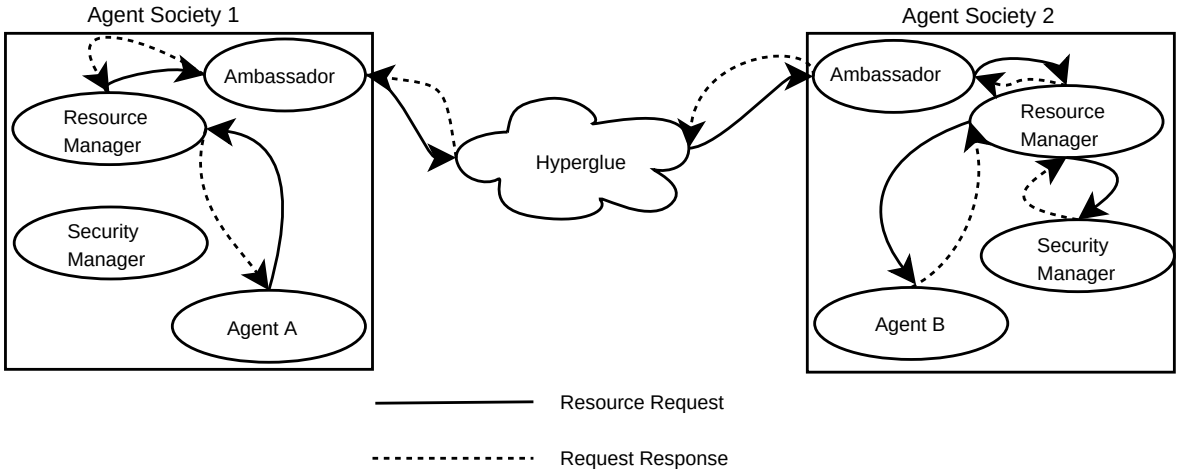

Figure 5-1: High-level overview of a inter-society resource request in Hyperglue

its own Resource Manager. The provider society's Resource Manager then queries its Security Manager to establish whether the resource request should be granted or not. If the Security Manager allows the release of the resource, then a handle to the resource is passed back via the two Ambassador agents.

5.4.2 Security Manager

In Hyperglue, a Security Manager is responsible for all security related decisions within an agent society. This means that it is responsible for such tasks as authentication and authorization. The Security Manager we implemented modularizes these tasks. Our implementation of the Security Manager takes in an agent request and passes it into both an Authentication module and an Access Control module. In our architecture these modules are pluggable and allow for extensibility and flexibility. If all the modules that the Security Manager relies on determine the agent request to be satisfiable, then the Security Manager informs the Resource Manager to proceed with the request. The Security Manager is invoked through the following interface:

```
public boolean isAllowed(Request req) throws RemoteException;
```

This allows the security mechanisms to be abstracted away from the other components of Hyperglue by a single simple interface.

5.4.3 Authentication Module

The Authentication Module we built was a placeholder reference implementation. Since the goal of this project was not to address authentication issues, our implementation assumes that all agent requests are authenticated by default. The Security Manager invokes the Authentication Module through the following interface:

```
public AuthenticatedRequestToken Authenticate(Request req) throws RemoteException;
```

We introduce a notion of an AuthenticatedRequestToken which contains elements in the request that the Authentication Module has validated. These include the name of the requesting society, the resource being requested, the validity of the request, and the name of the provider society (from whom the request is being made). These validated elements are later made available for use by other modules used by the Security Manager.

5.4.4 Access Control Module

This module acts as the bridge between ACCESS and Hyperglue. The Security Manager invokes the Access Control Module through the following interface:

```
public boolean isAllowed(AuthenticatedRequestToken art) throws RemoteException;
```

The Access Control module extracts the name of the resource being requested from the AuthenticatedRequestToken and retrieves a handle for the appropriate node in the semantic network describing the ACCESS policy. If the node is not already in memory a handle is obtained by instantiating it from the persistent storage. Thereafter, we evaluate the resource to determine if it can be released, and this causes the backward-chaining based inferencing described earlier to be initiated. The result is returned up to the Security Manager. If a node describing the resource being requested does not exist in the semantic network, no permissions are associated with that resource and it can be released at will. Therefore, the Security Manager is informed that the resource is available for release.

5.4.5 Accessing Dynamic Attributes

In the ACCESS ontology, the leaf nodes are able to obtain and use the current values of Attributes outside the scope of the semantic network. These nodes defined as ContextAttributes and EvidenceAttributes obtain live values from software agents within the same Hyperglue society that controls the resource being requested. We assert that when ACCESS is deployed in a Hyperglue environment, knowledge regarding attributes relevant within a society and taken into consideration when making access rights determinations exist in software agents running as part of that Hyperglue society. For example, a Calendar agent running within a person's society can provide attributes regarding the person's activities. Therefore, these particular nodes in the ACCESS ontology have been instrumented such that they are able to communicate with Hyperglue agents. However, agents are implemented over time and their interfaces and functionality generally varies, so we have defined an AttributeProvider interface that all agents providing information during ACCESS policy evaluations must implement.

At a very basic level, implementing the interface in a Hyperglue agent involves implementing a method of the following form:

```
public String attributeQuery(String method, String param) throws RemoteException;
```

Method is an identifier for the information being requested and *param* provides any information that may be required by the attribute provider. For example, we make information from the AuthenticatedRequestToken available for populating the *param* field. Thus, it is possible to define a policy where the EvidenceAttribute FacultyMember is determined by querying the FacultyDirectory agent with *method* IsMember and *param* REQUESTOR. By convention we use all uppercase tags to retrieve non-static values that need to be passed into such attributeQueries, and currently support the properties contained in the AuthenticatedRequestToken. Thus, at runtime, the above attributeQuery would occur with the *param* field populated with the requesting society name.

5.5 Example of Use

To demonstrate the effectiveness of ACCESS, we applied ACCESS on a toy application that retrieved contact information from other agent societies. That is, an application where if Alice wanted to contact Bob, an agent running in Alice's society would query a set of agents in Bob's society to determine his contact information. We are aware that this approach has a weakness with respect to time. That is, since we're simply controlling access to the contact information and not the contact channels themselves, once obtained it is possible for the contact information to be cached and used at the resource requester's discretion. This would circumvent the intent of the policies defined. Therefore, in an actual application of ACCESS we would secure the contact channels themselves.

However, for demonstration purposes, due to the unavailability of more sophisticated resources we used this approach and implemented a set of Contact Information Providers that can be instantiated by a given agent society. The need for supporting more complicated resources, and ACCESS's ability to support them is discussed further in Chapter 6. In this example case, the Contact Information Providers were intended for use by agent societies representing *Person* entities who worked in the Computer Science and Artificial Intelligence Laboratory at MIT.

Contact Information Providers

- *NonInteractiveContactInfoProvider* This agent can be queried for information about non-interactive channels for contacting the entity in which this agent has been instantiated. Thus, if this agent was instantiated in Bob's society it would contain knowledge about such elements as Bob's e-mail address, snail mail address, etc.

- *InteractiveContactInfoProvider* Contains Knowledge about interactive channels for contacting the entity in which this agent is instantiated. Such channels can include telephone number and instant messaging ID.

- *WalkingDirectionsInfoProvider* Given a starting location within the Stata Center, provides walking directions to the office of the person who's agent society this agent has been instantiated in. Walking directions are provided by leveraging work done for the LAIR project [28].

- *PresenceInfoProvider* An agent that can be queried with a location name to check if the person entity represented by the society this agent is running in is present at that location. This information can obtained by leveraging work done for the PLACE project [27].

The Contact Information Providers described here are limited in functionality and simple by design. They serve the purpose of demonstrating how ACCESS functions. One should note that more interesting resources from actual instant messaging channels to even real-time video conferencing channels can be implemented using the same agent framework. Furthermore, ACCESS will provide the same guarantees to applications of that nature.

Contact Information Retriever

We also built an application that retrieved and assembled contact information from another agent society for presentation to the user represented by a given agent society. This application was implemented as a ContactInformationRetriever agent. It would try to obtain handles for the four Contact Information Providers described above and query them for the knowledge contained within them. Thus, it could potentially obtain a diverse set of contact information elements about another entity.

Securing Contact Information Providers with ACCESS

Given this infrastructure, we formulated a sample ACCESS policy to be used in controlling access to the Contact Information Providers in a given agent society. This policy is given as a PolicyGen *policy file* in Appendix A. The policy we defined adds the following restrictions regarding how access permissions should be granted to Contact Information Providers.

- *NonInteractiveContactInfoProvider* In this case, we wanted to demonstrate how ACCESS would function when we want to place no restrictions on a resource. You will notice that the *policy file* makes no mention of it.

- *InteractiveContactInfoProvider* Here we wanted to show how ACCESS can be used to deal with instances where multiple conditions under which a resource can be released exist. In this case, we restricted access to the information under two different ReleaseConditions. We also wanted to demonstrate how restrictions on access rights can be made with respect to either requester roles, provider context, or a combination of both. The first ReleaseCondition requires the current time to be within the block of regular working hours (a ContextualState) and for the provider to be present in his office (another ContextualState). The second ReleaseCondition requires that requester is a CSAIL lab member (a RequesterRole) and that the provider be present in his office (a Contextual State).

- *WalkingDirectionsInfoProvider* For this case, we defined policies analogous to those defined for the InteractiveContactInfoProvider resource. The reason for doing this is to show that our design allows for the reuse of policy elements. If you examine the *policy file* in Appendix A, you will notice that both the InteractiveContactInfoProvider and the WalkingDirectionsInfoProvider share the same ReleaseConditions.

- *PresenceInfoProvider* We use this case to demonstrate how we still support the more conventional Role Based Access Control [40, 11] mechanisms. For this particular resource, we define permissions through a ReleaseCondition that just requires the requester to be a CSAIL lab member.

5.5.1 Inference Traces from ACCESS

To demonstrate how these resources are protected, a detailed and annotated trace of how ACCESS and Hyperglue function when working through this scenario is

provided in this section. Note that in the case that generated the trace, Bob is an entity trying to restrict access to his contact information and Alice is an entity trying to obtain his contact information. The trace demonstrates how a resource request transitions across Hyperglue from one society to another and is passed through its Ambassador agent, Resource Manager, Security Manager, Authentication Module, and Access Control Module before the request is fulfilled or denied. The trace that was run was configured to highlight the ACCESS system's functionality. We provide annotated traces for the process of obtaining handles on NonInteractiveContactInfo, InteractiveContactInfo and PresenceInfo resources. Since the trace for a WalkingDirectionsInfo resource is analogous to an InteractiveContactInfo resource we do not annotate it but provide it as Appendix B.

Obtaining a NonInteractiveContactInfo agent/resource

A request for the NonInteractiveContactInfo agent from Bob's society is initiated by Alice's society.

```
Alice:edu.mit.aire.hyperglue.Ambassador:
      Remote request for ag[Bob:occreq{demo.hyperglue.access.NonInteractiveContactInfo}]
```

The request transitions to Bob's society over Hyperglue, and results in the instantiation of Ambassador, SecurityManager, AuthenticationModule and AccessControlModule agents in Bob's society (since they hadn't been instantiated before). The request is also identified as being for Bob's society and therefore marked as a local request.

```
Bob:HYPERGLUE: Starting agent Bob:edu.mit.aire.hyperglue.Ambassador.
Bob:HYPERGLUE: Starting agent Bob:edu.mit.aire.hyperglue.security.SecurityManager.
Bob:HYPERGLUE: Starting agent Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule.
Bob:HYPERGLUE: Starting agent Bob:edu.mit.aire.hyperglue.security.access.AccessControlModule.
Bob:edu.mit.aire.hyperglue.Ambassador:
    Local request for ag[Bob:occreq{demo.hyperglue.access.NonInteractiveContactInfo}]
```

The AuthenticationModule retrieves relevant information from the request and validates it.

```
Bob:edu.mit.aire.hyperglue.security.SecurityManager: [ACCESS]Received a request for evaluation
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Received a request for Authentication
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
```

```
    [ACCESS]Request *IS* an AgentRequest
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Requestor Society:Alice
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule: [ACCESS]Provider Society:Bob
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Resource Occupation:demo.hyperglue.access.NonInteractiveContactInfo
```

The AccessControlModule (which is an implementation of ACCESS) realizes that no permissions are associated with the requested resources and allows it to be released. A handle to a NonInteractiveContactInfo agent is provided to Alice's society.

```
Bob:edu.mit.aire.hyperglue.security.access.AccessControlModule:
    [ACCESS]Received a request for Authorization
Bob:edu.mit.aire.hyperglue.security.access.AccessControlModule:
    [ACCESS]No permissions defined for this resource - allowing...
Alice:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.NonInteractiveContactInfo.
```

Obtaining an InteractiveContactInfo agent/resource

Alice's society makes a request for the InteractiveContactInfo resource in Bob's society.

```
Alice:edu.mit.aire.hyperglue.Ambassador:
                Remote request for ag[Bob:occreq{demo.hyperglue.access.InteractiveContactInfo}]
Bob:edu.mit.aire.hyperglue.Ambassador:
                Local request for ag[Bob:occreq{demo.hyperglue.access.InteractiveContactInfo}]
```

The request is passed into the SecurityManager which in turn hands it to the AuthenticationModule which retrieves relevant information from the request and validates it.

```
Bob:edu.mit.aire.hyperglue.security.SecurityManager: [ACCESS]Received a request for evaluation
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Received a request for Authentication
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Request *IS* an AgentRequest
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Requestor Society:Alice
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule: [ACCESS]Provider Society:Bob
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Resource Occupation:demo.hyperglue.access.InteractiveContactInfo
```

The SecurityManager next passes the request to the AccessControlModule that realizes there are two ReleaseConditions under which this resource can be released.

```
Bob:edu.mit.aire.hyperglue.security.access.AccessControlModule:
    [ACCESS]Received a request for Authorization
demo.hyperglue.access.InteractiveContactInfo: 2 Release Conditions Found.
```

The InOfficeDuringWorkingHours ReleaseCondition has no RoleRequirements so we assert that the RoleValdity is true. Next we need to evaluate the two Context-Conditions that need to be fulfilled for the InOfficeDuringWorkingHoursState to be applicable.

```
[ACCESS]Evaluating InOfficeDuringWorkHours Release Condition.
        No role requirements defined -> Role Validity is True
        InOfficeDuringWorkingHoursState: 2 Context Conditions Found.
```

We determine that there's one AttributeRequirement that needs to be fulfilled for the InOfficeCondition to be valid.

```
[ACCESS]Evaluating InOfficeCondition Context Condition.
        1 Attribute Requirements found.
```

The InOfficeRequirement requires the Attribute to be "true". It needs to be fulfilled for the InOfficeCondition to be valid.

```
[ACCESS]Evaluating InOfficeRequirement Attribute Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true
```

ACCESS evaluates the InOfficeAttribute by first instantiating the PresenceInfo agent in Bob's agent society and then making an attributeQuery on whether Bob is in his office. When this trace was obtained, he was in his office and therefore the attribute returns "true".

```
[ACCESS]Evaluating InOfficeAttribute Context Attribute.
Bob:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.PresenceInfo.
[ACCESS]Calling demo.hyperglue.access.PresenceInfo.attributeQuery(presence,office)
[ACCESS]InOfficeAttribute Returning:true
```

The result from the InOfficeAttribute means that both the InOfficeRequirement and InOfficeCondition hold true.

```
[ACCESS]InOfficeRequirement Returning:true
[ACCESS]InOfficeCondition Returning:true
```

ACCESS now evaluates the second ContextCondition, which is the condition that determines if the current time is within regular working hours.

```
[ACCESS]Evaluating WorkingHoursCondition Context Condition.
        1 Attribute Requirements found.
```

The WorkingHoursRequirement requires the Attribute to be "true". It needs to be fulfilled for the WorkingHoursCondition to be valid.

```
[ACCESS]Evaluating WorkingHoursRequirement Attribute Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true
```

Again, ACCESS needs to instantiate an Clock agent in Bob's society to evaluate the WorkingHoursAttribute. An attributeQuery on whether the current time is within the "WorkingHours" block is made. When this trace was obtained, the current time was outside this block and therefore the agent returns false.

```
[ACCESS]Evaluating WorkingHoursAttribute Context Attribute.
Alice:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.Clock.
[ACCESS]Calling demo.hyperglue.access.Clock.attributeQuery(InBlock,WorkingHours)
[ACCESS]WorkingHoursAttribute Returning:false
```

The result from the WorkingHoursAttribute means that both the WorkingHoursRequirement and WorkingHoursCondition are invalid.

```
[ACCESS]WorkingHoursRequirement Returning:false
[ACCESS]WorkingHoursCondition Returning:false
```

Since one of the required ContextConditions was invalid(ie. the WorkingHoursCondition), both the ContextualState encapsulating Bob being in his office during working hours and the ReleaseCondition associated with it are also invalid.

```
[ACCESS]InOfficeDuringWorkingHoursState Returning:false
[ACCESS]InOfficeDuringWorkHours Returning:false
```

ACCESS now tries to determine if the second ReleaseCondition is applicable. The InOfficeCSAILMemberRequest condition contains one RoleCondition and that will be evaluated first.

```
[ACCESS]Evaluating InOfficeCSAILMemberRequest Release Condition.
        CSAILMemberRole: 1 Role Conditions Found.
```

ACCESS tries to evaluate the RoleCondition determining whether the resource requester is a lab member and determines that it needs to have an EvidenceRequirement fulfilled first.

```
[ACCESS]Evaluating CSAILMemberCondition Role Condition.
        1 Evidence Requirements found.
```

The EvidenceRequirement asserts that it is only fulfilled when the attribute is equal to “true”.

```
[ACCESS]Evaluating CSAILMemberRequirement Evidence Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true
```

To determine the EvidenceAttribute, ACCESS has to instantiate a CSAILDirectory agent in Bob’s society. This agent interfaces with a central CSAIL directory to allow queries of interest to Bob. An attributeQuery to determine whether Alice is a member of CSAIL is executed and the result in this case turns out to be true.

```
[ACCESS]Evaluating CSAILMemberAttribute Evidence Attribute.
Bob:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.CSAILDirectory.
[ACCESS]Calling demo.hyperglue.access.FakeCSAILDirectory.attributeQuery(isMember,Alice)
[ACCESS]CSAILMemberAttribute Returning:true
```

Since the EvidenceAttribute is what was expected by the EvidenceRequirement and no other requirements are defined, the CSAILMemberRole is determined to be valid.

```
[ACCESS]CSAILMemberRequirement Returning:true
[ACCESS]CSAILMemberCondition Returning:true
[ACCESS]CSAILMemberRole Returning:true
```

ACCESS is now back to evaluating the ReleaseCondition and realizes that a ContextualState describing Bob’s presence which has an associated ContextCondition needs to be evaluated as well.

```
        InOfficeState: 1 Context Conditions Found.
```

The ContextCondition has one AttributeRequirement which needs to be valid.

```
[ACCESS]Evaluating InOfficeCondition Context Condition.
        1 Attribute Requirements found.
```

The AttributeRequirement asserts that it is only fulfilled when the attribute is equal to “true”.

```
[ACCESS]Evaluating InOfficeRequirement Attribute Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true
```

To determine the InOfficeAttribute, ACCESS makes an attributeQuery to the PresenceInfo agent (already instantiated earlier when evaluating the previous ReleaseCondition) to determine whether Bob is present in his office. At the moment that this trace was obtained, Bob was in his office and the InOfficeAttribute is determined to be true.

```
[ACCESS]Evaluating InOfficeAttribute Context Attribute.
[ACCESS]Calling demo.hyperglue.access.PresenceInfo.attributeQuery(presence,office)
[ACCESS]InOfficeAttribute Returning:true
```

Since no other constraints are specified, this result bubbles up to assert that the ContextualState representing Bob being in his office is valid.

```
[ACCESS]InOfficeRequirement Returning:true
[ACCESS]InOfficeCondition Returning:true
[ACCESS]InOfficeState Returning:true
```

Since both the RoleRequirement and ContextualState associated with the ReleaseCondition being evaluated are valid, ACCESS determines that the InteractiveContactInfo resource can be released and instantiates and provides a handle of it to Alice's society.

```
InOfficeCSAILMemberRequest Returning:true
demo.hyperglue.access.InteractiveContactInfo Returning:true
Alice:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.InteractiveContactInfo.
```

Obtaining a PresenceInfo agent/resource

Alice's society makes a request for the PresenceInfo resource in Bob's society.

```
Alice:edu.mit.aire.hyperglue.Ambassador:
      Remote request for ag[Bob:occreq{demo.hyperglue.access.PresenceInfo}]
Bob:edu.mit.aire.hyperglue.Ambassador:
    Local request for ag[Bob:occreq{demo.hyperglue.access.PresenceInfo}]
```

The request is passed into the SecurityManager which in turn first hands it to the AuthenticationModule which retrieves relevant information from the request and validates it.

```
Bob:edu.mit.aire.hyperglue.security.SecurityManager: [ACCESS]Received a request for evaluation
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Received a request for Authentication
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Request *IS* an AgentRequest
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Requestor Society:Alice
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule: [ACCESS]Provider Society:Bob
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Resource Occupation:demo.hyperglue.access.PresenceInfo
```

The SecurityManager next passes the request to the AccessControlModule which realizes there's just one ReleaseCondition under which this resource can be released.

```
Bob:edu.mit.aire.hyperglue.security.access.AccessControlModule:
    [ACCESS]Received a request for Authorization
demo.hyperglue.access.PresenceInfo: 1 Release Conditions Found.
```

ACCESS now tries to determine if the ReleaseCondition is applicable. The CSAILMemberRequest condition contains one RoleCondition and that will be evaluated first.

```
[ACCESS]Evaluating CSAILMemberRequest Release Condition.
        CSAILMemberRole: 1 Role Conditions Found.
```

ACCESS tries to evaluate the RoleCondition determining whether the resource requester is a lab member and determines that it needs to have an EvidenceRequirement fulfilled first.

```
[ACCESS]Evaluating CSAILMemberCondition Role Condition.
        1 Evidence Requirements found.
```

The AttributeRequirement asserts that it is only fulfilled when the attribute is equal to "true".

```
[ACCESS]Evaluating CSAILMemberRequirement Evidence Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true
```

To determine the EvidenceAttribute, ACCESS has to instantiate a CSAILDirectory agent in Bob's society. This agent interfaces with a central CSAIL directory to allow queries of interest to Bob. An attributeQuery to determine whether Alice is a member of CSAIL is executed and the result in this case turns out to be true.

```
[ACCESS]Evaluating CSAILMemberAttribute Evidence Attribute.
[ACCESS]Calling demo.hyperglue.access.CSAILDirectory.attributeQuery(isMember,Alice)
[ACCESS]CSAILMemberAttribute Returning:true
```

Since the EvidenceAttribute is what was expected by the EvidenceRequirement and no other requirements are defined, the CSAILMemberRole is determined to be valid.

```
[ACCESS]CSAILMemberRequirement Returning:true
[ACCESS]CSAILMemberCondition Returning:true
[ACCESS]CSAILMemberRole Returning:true
```

At the ReleaseCondition being evaluated we notice that no ContextRequirements are defined. Therefore, the resource can be released and we pass a handle to it back to Alice's society.

```
        No context requirements defined -> Context Validity is True


[ACCESS]CSAILMemberRequest Returning:true
demo.hyperglue.access.PresenceInfo Returning:true
Alice:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.PresenceInfo.
```

5.5.2 Discussion

This toy use case highlights some important capabilities in ACCESS. The policy we generated didn't have any references to the NonInteractiveContactInfoProvider. This is because ACCESS only models restrictions regarding when resources can be released. By doing so we reduce the complexity of policies that are created. In practice we have observed that only a subset of the total available resources actually need to be protected from unauthorized use. The ACCESS model allows administrators to focus on the resource elements that are important.

Furthermore, in defining policies for WalkingDirectionsInfoProvider and InteractiveContactInfoProvider we were able to model constraints using a mix of both resource provider context and resource requester role. In doing so, we have enabled the ACCESS policy to dynamically change and apply itself. Another feature in ACCESS illustrated by the permissions defined for these two resources is in the modularity and re-usability of policy elements. For example, at the lowest level the ContextAttributes and EvidenceAttributes used to represent time of day, presence and lab member status are represented as individual nodes in the semantic network describing the policy. They are used by ContextualState and RequesterRole nodes that represent "working hours," "in office," and "lab member". Those in turn are used by ReleaseConditions that describe conditions like "in office during working hours," "requester is a lab member," or "in office and requester is a lab member". Thus, we see that lower level nodes in the network are linked to and used by multiple nodes at higher levels. This means the representation of a single concept can

exist on a single node and be referenced by any other node. This also means that when the mechanics of that concept change, only one node needs to be changed to update the entire policy. For example, if the agent providing presence information changes, we simply need to update the ContextAttribute containing that information and the policy will continue to work as intended. This level of abstraction provides for greater maintainability of policies.

In restricting the PresenceInfoProvider to just lab members, we have shown that ACCESS doesn't add new constraints on existing access control mechanisms. In that particular case, we allowed permissions to be defined purely in terms of roles analogous to the role-based access controls currently popular. Therefore, ACCESS can be viewed as an evolution of existing access control mechanisms addressing new needs that have become apparent.

The next chapter provides an overview of the issues we expect to encounter in the future as ACCESS matures and sees more use.

Chapter 6

Future Issues

As Hyperglue evolves and sees extended use, we believe that ACCESS's role in Hyperglue enabled smart spaces will also increase. However, in order for that role to be an effective one a number of issues will need to be addressed first. In this chapter we identify some of these issues and discuss possible approaches that can be taken towards resolving them. In particular, the issues we focus on are:

- Policy definition mechanisms
- AI Planners
- System performance
- Modelling requester intent
- Authentication

Our existing mechanisms for defining ACCESS policies still leave much to be desired. While PolicyGen is an useful tool, it is far from being user friendly. Translating a semantic network describing an ACCESS policy into a form that PolicyGen can work with requires the ability to mentally visualize the network and the interdependencies within it. This is difficult for even the most sophisticated users. We believe a graphical tool that helps visualize the policy and allows its manipulation

and navigation will make this task significantly easier. This will make the function of different instances of each node type and their inter-dependencies clearer. Additionally, the tool could be enabled to check that the policy being defined is well-formed and point out potential errors as they are being defined. Furthermore, the ability to run sandboxed test cases on the policy being developed would also benefit the end user and increase the efficiency of the policy definition process.

As applications of smart spaces using Hyperglue increase, we expect there to be a greater level of interest in adding indirection. That is, resources like instant messaging channels, telephones, video feeds, etc. will be represented as Hyperglue resources. This enables those resources that were traditionally embedded independently in our environments to benefit from the facilities that Hyperglue affords. For example, an agent society representing Alice wanting to contact Bob could request a communication channel from Bob's society. Bob's agent society would make available a channel that took into account his preferences for communication. But let us consider a case where Alice required a particular type of communication channel– a video conferencing channel since she needed to show a diagram to Bob. It may be that Bob's ACCESS policies don't allow the release of this resource at that moment. Alice would have to find alternative means to achieve her goal. Perhaps she may have to send the diagram via e-mail. This approach does not scale well as the number of options available grow. Instead, it would be desirable for ACCESS to support planners[16] running on other agent societies. Thus, they would be able to determine a-priori the likelihood that a given agent society will release a desired resource. This enables the planner to generate and execute plans that have a higher likelihood of succeeding. A caveat: if the manner in which policy evaluations are done become exposed to external entities, one becomes susceptible to exploitation via exploratory or probing attacks. In the case of an ill-defined policy this may allow security compromises to occur. Given that we can reasonably predict what normal behavior is, we believe it is possible to detect such attacks when they are occurring (since they will differ from the norm). Therefore, taking remedial action at that time is an option that can mitigate this risk.

For broader use, ACCESS will also have to make some performance enhancements. Currently we evaluate policies in a naive manner where nodes in the semantic network describing policies are retrieved from persistence storage and instantiated into memory as and when needed. However, no garbage collection occurs and as policies grow larger virtual memory and the performance issues associated with it will come into play. If the relevant permissions associated with an ACCESS policy cannot be evaluated within a reasonable time frame, ACCESS will hinder rather than enhance the user experience. We believe this issue can be addressed with smarter caching mechanisms where only node objects that are frequently required are kept in memory. Furthermore, the retrieval of attribute values can also delay policy evaluations. We propose to add functionality that allows administrators to define how long an attribute remains valid. Thus, attribute values that are less dynamic can be cached until they are stale. This will reduce the overhead of having to retrieve attribute values from the appropriate attribute provider during every evaluation.

We believe it's also important to support ACCESS policy evaluations where intended use is taken into account. That is, we might want to allow the release of certain resources if the requester has a specific use for them in mind that doesn't conflict with the provider's preferences. For example, if the request is for a communication channel via telephone and the intended use is to provide information about an emergency relevant to the provider, we would like to allow the request to occur. Unfortunately, in our existing framework of ACCESS we're unable to trust context attributes provided by the resource requester. So, the requester has no trusted mechanism for letting the provider know about the intended use. A broader mechanism that allows trust to be placed on external context providers needs to be established. This could occur as part of the support for an authentication framework we desire and discuss below. Furthermore, Hyperglue's internal request marshalling mechanism needs to be extended to support the inclusion of additional data. Given these, the amount of interesting applications that can be supported by Hyperglue using ACCESS would increase significantly.

Finally, we believe there is a need for an authentication framework complimentary to ACCESS. While ACCESS currently provides a framework for establishing access control policies in Hyperglue-enabled applications, it inherently relies on the resource requests being authenticated. Thus, without a complimentary authentication mechanism, ACCESS's effectiveness is questionable. The society-based organization imposed in Hyperglue has strong synergies with domains where the widely used Public Key Infrastructure[12] for authentication, privacy, and non-repudiation is usually applied. It is feasible to establish a trusted authority that can validate and issue identifying tokens as X.509 certificates to an agent society when it is established. These tokens can be used for establishing trust amongst agent societies.

We believe these are the primary issues of concern as we move forward with ACCESS and its use in Hyperglue. In the next chapter we will provide some concluding thoughts.

Chapter 7

Conclusion

In this research project we have highlighted security and privacy issues related to smart spaces. In particular, we focused attention on the access control mechanisms required in such applications. In doing so, we have argued for and proposed a set of requirements for access control frameworks in smart spaces. Based on these requirements, we designed ACCESS (Access Controls for Cooperatively Enabled Smart Spaces); a framework for access control specification, evaluation, and enforcement. ACCESS takes into consideration both resource provider context and resource requester roles in determining access rights. This enables dynamic context to be factored in when making access control decisions. Since smart spaces are dynamic in nature, we believe our system avoids some of the pitfalls in other approaches taken in the same domain.

We have also built a reference implementation of ACCESS for use within our own Hyperglue framework. Our implementation includes an ontology for representing access control policy information. We instantiate policies in a semantic network where we can perform reasoning. We have extended the SEMANTIC framework to support the use of context cues obtained from Hyperglue-enabled agents when performing reasoning on a semantic network. Thus, we are able to perform reasoning on dynamic data. Furthermore, we have taken a novel approach in embedding the inferencing mechanisms within the knowledge representation itself. While this is an artifact of the object-oriented nature of our knowledge representation, it has

eliminated the need for an external inferencing engine, thereby allowing greater flexibility and extensibility in the inferencing mechanisms.

To assist end-users in defining relevant policies we have defined a simple yet expressive declarative policy definition language. A compiler that parses this policy definition language to produce a corresponding semantic network which can be instantiated in SEMANTIC was also implemented. We believe that this policy definition language provides the basic building blocks on which abstractions allowing for greater usability can be built.

In conclusion, our work has shown that dynamic, role-based, context-aware access controls can be a powerful means to enable the types of access controls we desire and require in smart spaces. We have also shown that such access control mechanisms, while supporting complex policies, aren't unwieldy but rather have practical use. We believe that as Hyperglue-based applications continue to be deployed, ACCESS will provide a critical set of features required for their success.

Appendix A

PolicyGen: Sample ACCESS Policy

```
;;FILENAME:access-demo.pol
;;
;;ACCESS Policy Definition
;;creates an instance of a policy based on the ACCESS policy definition ontology
;;
;;syntax uses 3 primitives:
;;1. is-a <instance_name> <node_type>
;;   creates an instance of a node
;;   ex. "is-a WeatherAgent Resource"
;;
;;2. property-value <instance_name> <property_name> <value>
;;   adds a property value to the instance
;;   ex. "property-value WeatherAgent AgentID edu.mit.aire.applications.info.weather.WeatherDisplay"
;;
;;3. property-value-list <instance_name> <property_name> <value list>
;;   adds a property value list to an instance. that is, a property that is a list
;;   can be populated using this mechanism.
;;   ex. "property-value-list WeatherAgent ReleaseIf relcond1, relcond2, relcond3"
;;
;; ACCESS Demo Policy


;;Evidence Attribute
is-a CSAILMemberAttribute EvidenceAttribute

;;Context Attribute
is-a InOfficeAttribute ContextAttribute
is-a WorkingHoursAttribute ContextAttribute

;;EvidenceRequirement
```

```
is-a CSAILMemberRequirement EvidenceRequirement

;;Attribute Requirement
is-a InOfficeRequirement AttributeRequirement
is-a WorkingHoursRequirement AttributeRequirement

;;Role Condition
is-a CSAILMemberCondition RoleCondition

;;Context Conditions
is-a InOfficeCondition ContextCondition
is-a WorkingHoursCondition ContextCondition

;;Requestor Roles
is-a CSAILMemberRole RequestorRole

;;ContextualStates
is-a InOfficeState ContextualState
is-a WorkingHoursState ContextualState
is-a InOfficeDuringWorkingHoursState ContextualState

;; ReleaseConditions
is-a InOfficeDuringWorkHours ReleaseCondition
is-a InOfficeCSAILMemberRequest ReleaseCondition
is-a CSAILMemberRequest ReleaseCondition

;; Resources
is-a InteractiveContactInfo Resource
is-a WalkingDirections Resource
is-a PresenceInfo Resource

;;CSAILMemberAttribute
property-value CSAILMemberAttribute Name "CSAILMemberAttribute"
property-value CSAILMemberAttribute EvidenceAgentID "demo.hyperglue.access.CSAILDirectory"
property-value CSAILMemberAttribute EvidenceQuery "isMember_REQUESTOR"

;;InOfficeAttribue
property-value InOfficeAttribute Name "InOfficeAttribute"
property-value InOfficeAttribute ContextAgentID "demo.hyperglue.access.PresenceInfo"
property-value InOfficeAttribute ContextQuery "presence_office"

;;WorkingHoursAttribute
property-value WorkingHoursAttribute Name "WorkingHoursAttribute"
property-value WorkingHoursAttribute ContextAgentID "demo.hyperglue.access.Clock"
property-value WorkingHoursAttribute ContextQuery "InBlock_WorkingHours"
```

```
;;CSAILMemberRequirement
property-value CSAILMemberRequirement Name "CSAILMemberRequirement"
property-value CSAILMemberRequirement FulfillmentCheckType "="
property-value CSAILMemberRequirement FulfilledWhen "true"
property-value CSAILMemberRequirement Attribute CSAILMemberAttribute

;;InOfficeRequirement
property-value InOfficeRequirement Name "InOfficeRequirement"
property-value InOfficeRequirement FulfillmentCheckType "="
property-value InOfficeRequirement FulfilledWhen "true"
property-value InOfficeRequirement Attribute InOfficeAttribute

;;WorkingHoursRequirement
property-value WorkingHoursRequirement Name "WorkingHoursRequirement"
property-value WorkingHoursRequirement FulfillmentCheckType "="
property-value WorkingHoursRequirement FulfilledWhen "true"
property-value WorkingHoursRequirement Attribute WorkingHoursAttribute

;;CSAILMemberCondition
property-value CSAILMemberCondition Name "CSAILMemberCondition"
property-value-list CSAILMemberCondition EvidenceRequirements CSAILMemberRequirement

;;InOfficeCondition
property-value InOfficeCondition Name "InOfficeCondition"
property-value-list InOfficeCondition AttributeRequirements InOfficeRequirement

;;WorkingHoursCondition
property-value WorkingHoursCondition Name "WorkingHoursCondition"
property-value-list WorkingHoursCondition AttributeRequirements WorkingHoursRequirement

;;CSAILMemberRole
property-value CSAILMemberRole Name "CSAILMemberRole"
property-value-list CSAILMemberRole ValidIf CSAILMemberCondition

;;InOfficeState
property-value InOfficeState Name "InOfficeState"
property-value-list InOfficeState ValidIf InOfficeCondition

;;WorkingHoursState
property-value WorkingHoursState Name "WorkingHoursState"
property-value-list WorkingHoursState ValidIf WorkingHoursCondition

;;InOfficeDuringWorkingHoursState
property-value InOfficeDuringWorkingHoursState Name "InOfficeDuringWorkingHoursState"
```

```
property-value-list InOfficeDuringWorkingHoursState ValidIf InOfficeCondition, WorkingHoursCondition

;;InOfficeDuringWorkHours
property-value InOfficeDuringWorkHours Name "InOfficeDuringWorkHours"
property-value InOfficeDuringWorkHours Context InOfficeDuringWorkingHoursState

;;InOfficeCSAILMemberRequest
property-value InOfficeCSAILMemberRequest Name "InOfficeCSAILMemberRequest"
property-value InOfficeCSAILMemberRequest Context InOfficeState
property-value InOfficeCSAILMemberRequest Role CSAILMemberRole

;;CSAILMemberRequest
property-value CSAILMemberRequest Name "CSAILMemberRequest"
property-value CSAILMemberRequest Role CSAILMemberRole

;;InteractiveContactInfo
property-value InteractiveContactInfo Name "InteractiveContactInfo"
property-value InteractiveContactInfo Society "Bob"
property-value InteractiveContactInfo AgentID "demo.hyperglue.access.InteractiveContactInfo"
property-value-list InteractiveContactInfo ReleaseIf InOfficeDuringWorkHours, InOfficeCSAILMemberRequest

;;WalkingDirections
property-value WalkingDirections Name "WalkingDirections"
property-value WalkingDirections Society "Bob"
property-value WalkingDirections AgentID "demo.hyperglue.access.WalkingDirectionsInfo"
property-value-list WalkingDirections ReleaseIf InOfficeDuringWorkHours, InOfficeCSAILMemberRequest

;;PresenceInfo
property-value PresenceInfo Name "PresenceInfo"
property-value PresenceInfo Society "Bob"
property-value PresenceInfo AgentID "demo.hyperglue.access.PresenceInfo"
property-value-list PresenceInfo ReleaseIf CSAILMemberRequest
```

Appendix B

Obtaining a WalkingDirectionsInfo agent/resource

We apply the same restrictions applied to the InteractiveContactInfo agent/resource to the WalkingDirectionsInfo agent/resource. This trace is analogous to that shown above and therefore we have not repeated the annotations here.

```
Alice:edu.mit.aire.hyperglue.Ambassador:
      Remote request for ag[Bob:occreq{demo.hyperglue.access.WalkingDirectionsInfo}]

Bob:edu.mit.aire.hyperglue.Ambassador:
    Local request for ag[Bob:occreq{demo.hyperglue.access.WalkingDirectionsInfo}]

Bob:edu.mit.aire.hyperglue.security.SecurityManager: [ACCESS]Received a request for evaluation
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Received a request for Authentication
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Request *IS* an AgentRequest
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Requestor Society:Alice
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule: [ACCESS]Provider Society:Bob
Bob:edu.mit.aire.hyperglue.security.authentication.AuthenticationModule:
    [ACCESS]Resource Occupation:demo.hyperglue.access.WalkingDirectionsInfo

Bob:edu.mit.aire.hyperglue.security.access.AccessControlModule:
    [ACCESS]Received a request for Authorization
demo.hyperglue.access.WalkingDirectionsInfo: 2 Release Conditions Found.

[ACCESS]Evaluating InOfficeDuringWorkHours Release Condition.
        No role requirements defined -> Role Validity is True
        InOfficeDuringWorkingHoursState: 2 Context Conditions Found.
```

```
[ACCESS]Evaluating InOfficeCondition Context Condition.
        1 Attribute Requirements found.

[ACCESS]Evaluating InOfficeRequirement Attribute Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true

[ACCESS]Evaluating InOfficeAttribute Context Attribute.
[ACCESS]Calling demo.hyperglue.access.PresenceInfo.attributeQuery(presence,office)
[ACCESS]InOfficeAttribute Returning:true

[ACCESS]InOfficeRequirement Returning:true
[ACCESS]InOfficeCondition Returning:true

[ACCESS]Evaluating WorkingHoursCondition Context Condition.
        1 Attribute Requirements found.

[ACCESS]Evaluating WorkingHoursRequirement Attribute Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true

[ACCESS]Evaluating WorkingHoursAttribute Context Attribute.
[ACCESS]Calling demo.hyperglue.access.Clock.attributeQuery(InBlock,WorkingHours)
[ACCESS]WorkingHoursAttribute Returning:false

[ACCESS]WorkingHoursRequirement Returning:false
[ACCESS]WorkingHoursCondition Returning:false
[ACCESS]InOfficeDuringWorkingHoursState Returning:false
[ACCESS]InOfficeDuringWorkHours Returning:false

[ACCESS]Evaluating InOfficeCSAILMemberRequest Release Condition.
        CSAILMemberRole: 1 Role Conditions Found.

[ACCESS]Evaluating CSAILMemberCondition Role Condition.
        1 Evidence Requirements found.

[ACCESS]Evaluating CSAILMemberRequirement Evidence Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true

[ACCESS]Evaluating CSAILMemberAttribute Evidence Attribute.
[ACCESS]Calling demo.hyperglue.access.FakeCSAILDirectory.attributeQuery(isMember,Alice)
[ACCESS]CSAILMemberAttribute Returning:true
```

```
[ACCESS]CSAILMemberRequirement Returning:true
[ACCESS]CSAILMemberCondition Returning:true
[ACCESS]CSAILMemberRole Returning:true

        InOfficeState: 1 Context Conditions Found.

[ACCESS]Evaluating InOfficeCondition Context Condition.
        1 Attribute Requirements found.

[ACCESS]Evaluating InOfficeRequirement Attribute Requirement.
        FulfillmentCheckType is =
        FulfilledWhen is true

[ACCESS]Evaluating InOfficeAttribute Context Attribute.
[ACCESS]Calling demo.hyperglue.access.PresenceInfo.attributeQuery(presence,office)
[ACCESS]InOfficeAttribute Returning:true

[ACCESS]InOfficeRequirement Returning:true
[ACCESS]InOfficeCondition Returning:true
[ACCESS]InOfficeState Returning:true

[ACCESS]InOfficeCSAILMemberRequest Returning:true
demo.hyperglue.access.WalkingDirectionsInfo Returning:true
Alice:HYPERGLUE: Starting agent Bob:demo.hyperglue.access.WalkingDirectionsInfo.
```

Bibliography

[1] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and M. Dennis Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom '03)*, pages 52–58, Dallas-Fort Worth, Texas, USA, mar 2003. IEEE.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[3] Rodney Brooks. The intelligent room project. In *Proceedings of the 2nd International Cognitive Technology Conference (CT'97)*, Aizu, Japan, 1997.

[4] Roy H. Campbell, Jalal Al-Muhtadi, Prasad Naldurg, Geetanjali Sampemane, and M. Dennis Mickunas. Towards security and privacy for pervasive computing. In M. Okada, B. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, editors, *Software Security – Theories and Systems*, number XI in Lecture Notes in Computer Science, pages 1–15. Springer-Verlag, 2003.

[5] Edgar Frank Codd. Relational database: a practical foundation for productivity. *Communications of the ACM*, 25(2):109–117, 1982.

[6] Michael Coen. Sodabot: A software agent environment and construction system. Technical report, MIT AI Lab, jun 1994.

[7] Michael Coen. Building brains for rooms: Designing distributed software agents. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence. (IAAI97)*, 1997.

[8] Michael Coen, Henry Kautz, Bart Selman, Steven Ketchpel, and Chris Ramming. An experiment in the design of software agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, 1994.

[9] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.

[10] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20. ACM Press, 2001.

[11] David Ferraiolo and Rick Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[12] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194. Springer-Verlag, 1987.

[13] Krysztof Gajos and Ajay Kulkarni. FIRE: An Information Retrieval Interface For Intelligent Environments. In *Proceedings of International Workshop on Information Presentation and Natural Multimodal Dialogue (IPNMD 2001)*, dec 2001.

[14] Simon Godik and Tim Moses. eXtensible Access Control Markup Language (XACML) Version 1.0. Technical report, OASIS, feb 2003.

[15] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification, Second Edition*. The Java Series. Addison-Wesley, 2000.

[16] Cordell Green. Application of theorem proving to problem solving. Technical Report 4, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Mar 1969. Project 7494 Presented at IJCAI 1969.

[17] Joint Picture Expert Group. *Digital Compression and Coding of Continuous-tone Still Images, ISO/IEC 10918-1*. Geneva, Switzerland, 1994.

[18] Object Management Group. CORBA/IIOP Specification, mar 2004.

[19] Tracy Hammond, Krysztof Gajos, Randall Davis, and Howard Shrobe. An Agent-Based System for Capturing and Indexing Software Design Meetings. In *Proceedings of the International Workshop on Agents in Design (WAID'02)*, aug 2002.

[20] Nicholas Hanssens, Ajay Kulkarni, Rattapoom Tuchinda, and Tyler Horton. Building agent-based intelligent workspaces. In *ABA Conference Proceedings*, jun 2002.

[21] Brad Johanson and Armando Fox. Extending tuplespaces for coordination in interactive workspaces. *J. Syst. Softw.*, 69(3):243–266, 2004.

[22] Brad Johanson, Armando Fox, and Terry Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, pages 67–74, apr 2002.

[23] Lalana Kagal, Tim Finin, and Anupam Joshi. Declarative policies for describing web service capabilities and constraints. In *W3C Workshop on Constraints and Capabilities for Web SErvices*, Redwood Shores, CA, oct 2004.

[24] Max Van Kleek. Computation for the corridor: Experiments in augmenting public spaces. To be published.

[25] Max Van Kleek. Intelligent Environments for Informal Public Spaces: the Ki/o Kiosk Platform. Master's thesis, MIT, feb 2003.

[26] Arun Kumar, Neeran Karnik, and Girish Chafle. Context sensitivity in role-based access control. *SIGOPS Oper. Syst. Rev.*, 36(3):53–66, 2002.

[27] Justin Lin, Robert Laddaga, and Hirohisa Naito. Personal location agent for communicating entities (PLACE). In *4th International Symposium on Human Computer Interaction with Mobile Devices*, Pisa, Italy, 2002.

[28] Gary Look, Buddhika Kottahachchi, Robert Laddaga, and Howard Shrobe. A location representation for generating descriptive walking directions. In *IUI 2005: International Conference on Intelligent User Interfaces*, San Diego, CA, USA, Jan 2005. ACM.

[29] Gordon Moore. Cramming more components onto integrated circuits. *Electronics*, pages 114–117, apr 1965.

[30] Oxygen: Pervasive, human-centered computing. MIT Laboratory of Computer Science, Cambridge, Massachusetts, USA, may 2002.

[31] Stephen Peters, Gary Look, Kevin Quigley, Howard Shrobe, and Krzysztof Gajos. Hyperglue: Designing high-level agent communication for distributed applications. Originally submitted to AAMAS'03.

[32] Stephen Peters and Howie Shrobe. Using semantic networks for knowledge representation in an intelligent environment. In *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*, Ft. Worth, TX, USA, March 2003. IEEE.

[33] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. Icrafter: A service framework for ubiquitous computing environments. In *Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 56–75. Springer-Verlag, 2001.

[34] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iros. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom '03)*, pages 52–58, Dallas-Fort Worth, Texas, USA, mar 2003. IEEE.

[35] Eric Rescorla. SSL and TLS: Designing and building secure systems. Addison-Wesley, 2000.

[36] Manuel Romn, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, pages 74–83, oct 2002.

[37] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, section 9.4, pages 287–295. Prentice Hall Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, New Jersey, second edition, 2003.

[38] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, section 10.6, pages 349–354. Prentice Hall Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, New Jersey, second edition, 2003.

[39] Geetanjali Sampemane, Prasad Naldurg, and Roy H. Campbell. Access control for active spaces. In *Proceedings of the 18th Annual Computer Security Applications Conference*, page 343. IEEE Computer Society, 2002.

[40] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[41] Yee Jiun Song, Wendy Tobagus, Der Yao Leong, Brad Johanson, and Armando Fox. isecurity: A security framework for interactive workspaces. Technical report, Stanford University, 2003.

[42] Norbert A. Streitz, Jorg Geibler, Torsten Holmer, Shin'ichi Konomi, Christian Muller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-land: an interactive landscape for creativity and innovation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 120–127. ACM Press, 1999.

[43] Doug Sutherland. Rmi and java distributed computing. Technical report, Sun Microsystems, Inc., Nov 1997.

[44] Mark Torrance. Advances in human-computer interaction: The intelligent room. In *Working Notes of the CHI 95 Research Symposium*, Aizu, Japan, may 1995.

[45] Andrzej Uszok, Jeffrey M. Bradshaw, Matthew Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and Sturat Aiken. Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.

[46] W3C. *OWL Web Ontology Language*. http://www.w3.org/TR/owl-features/.

[47] W3C. *Resource Description Framework*. http://www.w3.org/RDF/.

[48] Mark Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–104, sep 1991.

[49] Mark Weiser and John Seely Brown. The coming age of calm technology. Technical report, Xerox PARC, 1996.

[50] Recommendation x.509: The directory-authentication framework. CCITT, 1988.