Policy-Guided Interactions in Ubiquitous Computing Systems

Ph.D. Dissertation Prospectus

V. Ramakrishna

vrama@cs.ucla.edu

Advisor: Dr. Peter Reiher

Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA

Abstract

The ubiquitous computing vision that promises access to information, personal data, computing resources and network connectivity everywhere and at any time will soon be a reality. In the practical manifestation of this vision, both the infrastructure and personal mobile devices accompanying human users will play a part. Device, network and application characteristics will become increasingly heterogeneous, and computing devices will be operating in a highly dynamic environment. Not only must a device respond to its user's needs in a context-aware manner, it must also ensure that security and privacy considerations are met. Additionally, it is impractical to expect mostly naïve users to constantly reconfigure their systems to obtain connections or to frequently change security and privacy settings to enable interoperation with other devices and domains.

This prospectus will show how devices and networks can interoperate on the basis of local private policies that collectively describe user preferences and security and resource constraints. We will describe a general purpose framework that allows these computers to *negotiate* among themselves using these policies, which would be specified in a flexible and expressive language with domain independent semantics. Negotiation will not require that the devices know each other's identity or have any prior trust relationship. Security and access control requirements will be met by building this framework on top of a sound trust model. Resource accesses can be negotiated on the basis of local measures of utility and risk-benefit assessment using the trust model. The framework will adapt to changing context, and will allow dynamic policy changes, making it more useful than an application written for the sole purpose of maintaining a particular set of policy constraints. Not only will this solution scale and allow decentralized control, it will also give administrators and users the freedom to configure the devices and resources under their control without having to worry unduly about interoperation and access control. This policy management and negotiation framework will be built as part of the Panoply [Eustice2003a] ubiquitous computing infrastructure project and will be evaluated using a varied set of application scenarios and policies.

Contents

Figures and Tables

1. Introduction

The current state-of-the-art technology in computing enables static desktop computing and occasional mobile computing in familiar environments. The transition to ubiquitous computing presents unique challenges to researchers. Advances in technology have given us powerful computers, both static and mobile, system software and applications that increase in *intelligence* and usability by leaps and bounds, and communication technologies, both wired and wireless, which are increasingly pervasive and provide easier and more efficient ways of connecting. Even with computers and networks pervading more of the physical space around us, ordinary users do not often get optimal security and usability with existing solutions. Typical environments where users need service are either completely open or are extremely picky about the devices with which they interact, and have stringent security policies that infringe on the privacy of the user devices (which must agree to the policies to obtain service). These devices and networks should not be, and do not have to be, so rigid in their interaction mechanisms. *Negotiations* between devices and networks could allow them to interact in ways that produce agreeable results in terms of their resource needs, and their privacy and security; automated protocols can produce such optimal results. This prospectus outlines the problems involved in negotiation, and the system research that will be necessary in order to obtain solutions. The remainder of this section will describe the ubiquitous computing vision, and elaborate on the problems that we are dealing with.

1.1. Toward a Ubiquitous Computing Future

Dr. Mark Weiser's 1991 vision of ubiquitous computing (*ubicomp* for short) [Weiser1991] is a futuristic vision (which gets more realistic every day) in which computers pervade the physical space around us without being obvious to the human eye. Such an infrastructure will help users perform various tasks, offer useful services and allow information access anywhere and at any time. More recently, Kindberg and Fox [Kindberg2002] identified physical integration and spontaneous interoperation as the two key all-encompassing characteristics of ubicomp systems.

Significant progress has been made in the physical integration area thanks to embedded systems research. Not only can our mobile accessories, like cell phones, PDAs and watches, run complex applications, offer services like GPS, and communicate wirelessly using embedded processors, but such capabilities can be embedded even into our refrigerators, walls and clothing in the near future using sensors, actuators and interfaces. Ubiquitous networking, another prerequisite for ubicomp, is rapidly becoming a reality with the effort to produce standards for wireless MANs, or WiMax [IEEE802.16] and vehicular networking in addition to the already established standards for wired and wireless LANs, cellular, satellite and personal area networks.

Smart space projects like Oxygen [MIT-Oxygen], Gaia [Román2002], One.world [Grimm2004a][Grimm2004b] and Centaurus [Kagal2001a][Kagal2001b][Undercoffer2003] propose infrastructure designs to dynamically manage resources, enable seamless communication, and adapt to changing context using a mixture of physically integrated components and mobile devices. Such designs are meant to manage local *hot spots* and do not scale to a global distributed system. Increasing physical integration will ensure ubiquity of such spots, or spaces. Local interactions between users' personal devices and local domains (or hot spots), which are interconnected through a global internet, are sufficient for user requirements. Therefore, to be able to compute and network ubiquitously, devices and smart spaces must have the capability for interoperating spontaneously unless explicitly prohibited by their managers. A primary obstacle is the heterogeneity of hardware, software, resources, networking technologies and applications that each local domain will possess. The systems mentioned above rely on

standardization of these features for global interoperation, which is practically not enforceable. Even if certain mechanisms, hardware, system software and networking technologies become *de* facto standards through popularity or market forces (like TCP/IP networking culminating in the Internet and the World Wide Web), different people manage and use local domains, and have different needs and expectations. They might use the design and resource management principles proposed by Oxygen or Centaurus, but they will have unique resource requirements and expect certain functionality from their systems, in addition to having unique security and privacy requirements. The infrastructure deployed at physical spaces that will enable ubicomp is growing in a decentralized fashion through the efforts of research groups and companies. Though it will create the problems described above, bottom-up growth of the infrastructure is both inevitable and desirable as it promotes innovation and lets designers and administrators make independent choices. The problem is to enable a standard through which different devices and networks can interoperate although they are managed and used by different people, do not share resource capabilities, security and privacy constraints, or have pre-decided trust relationships.

1.2. Interoperability

Interoperation among devices refers to the ability to communicate information about objects, offer services, discover external services, and access external resources through suitable interfaces. Users will expect to obtain and make use of resources wherever they go and to run their applications and obtain private data and information, either through their personal devices or local clients. Our next key challenge in the ubiquitous computing arena is to ensure spontaneous interoperation among any set of devices and administrative domains, typically between user devices and wireless networks, though occasionally also between two devices that are not necessarily mobile. Current approaches toward interoperation suffer from extremes. One approach advocates rigid policies governing interactions, which typically limit such interactions to devices or domains which have prearranged trust relationships, and/or predeployed mechanisms to make use of external services when two devices/networks come into contact. At the other extreme, systems use standard open interfaces which allow free interoperation at the expense of security and privacy (more about this in Section 1.3). This is the design approach of projects like Oxygen's Metaglue [Coen1999] and Sun's JINI [Waldo1999] which enable resource discovery and access. These and other ubicomp infrastructures, until recently, have not considered user and device mobility. Neither do they consider the dynamism of the environment, where interaction mechanisms, goals and constraints must vary with context. They also assume that mobile devices have similar capabilities, and that physical spaces contain uniform hardware, applications and interfaces, which is impractical (as mentioned earlier).

A body of research exists for interoperation at lower layers, and fairly standard spontaneous networking techniques are in use today. Interoperation in a mobile computing environment is made easier using techniques like mobile IP [Bhagwat1996] and application session handoff frameworks like IMASH [Bagrodia2003]. The *grid* [GRID] enables resource sharing in a large-scale distributed environment, and has only recently considered mobility and pervasive services [Bruneo2003][Anius2003], with emphasis on QoS, service discovery and task scheduling rather than security. The interoperation we are talking about in this prospectus refers to application or middleware level interoperation. Different computing entities need a common semantic framework to describe all kinds of information, resources and protocols; this is an ongoing research effort in the *semantic web* [SemWeb] project. These entities also need to be able to reconcile their resource needs, capabilities and security policies in order to interoperate, and the research proposed here will attempt to fill that gap.

1.3. Security and Privacy

Security and privacy are extremely important in any multi-user system, and especially so in an open ubiquitous computing environment. Permitting external entities to discover local resources and to make use of them raises privacy and security concerns, and this must be balanced against the necessity and usefulness of such access. Security and protection of private resources must be integrated into the interaction mechanisms and interfaces. The traditional approach of building mechanisms and interfaces and adding security support later does not work; dynamic resource allocation and sharing could result in unforeseen side effects and security holes. Using rigid security policies and access control rules to protect networks and the devices and resources within prevents interactions with unknown devices and those with incompatible configurations. Likewise, a user device must protect itself from potentially malicious networks through the maintenance of its own security policies. Still, in ubicomp, one cannot completely specify security policy, since it is impossible to anticipate every possible interaction with every possible computing entity beforehand. Therefore, to preserve the fundamentally open nature of ubicomp without giving up on security, autonomous systems must have the capability to make intelligent tradeoffs in particular contexts when deciding the nature and scope of interactions, based on a limited set of policies.

Systems can use a wide variety of security enforcement mechanisms like virus scanners, firewalls, tools like *nmap*, intrusion detection utilities, and QED [Eustice2003b], but these must be augmented with security policies that dictate how and when such mechanisms should be used. Similarly, existing frameworks for protection of privacy and access control, like *Kerberos*, access control lists, capabilities and role-based access control frameworks, work only through enforcement of rigid rules, and in addition, suffer from lack of scalability. More powerful access control models like GRBAC [Covington2000] and DRBAC [Freudenthal2002] also impose limitations which make them directly usable only in certain situations, conditions and environments. The notion of trust models is still evolving, and will form a key backbone for any ubicomp system although the current certificate-driven approach requires common trust authorities. As this prospectus will show, we can enable flexible interoperation through the use of policies that specify how and when such mechanisms can be used.

1.4. Usability and Interfaces

A ubiquitous computing system must be *usable*, and provide *intuitive* ways to let non-technical users interface with their personal devices and with the invisible infrastructure components in the background. It would be impractical to expect such users to change configuration settings in devices to establish network connectivity, change security and privacy levels, or explicitly run commands to discover and obtain resources. Users and system administrators should be able to set policy and expect their devices to figure out which lower-level resources, like network connectivity, display and audio devices, file systems, and so on are required and then obtain these from any environment they happen to be in. Devices should also adjust service and information provided to users with context, based on user policy. They should be more flexible in balancing security and resource needs, without requiring users to make arbitrary decisions whenever the primary goal results in failure. Frameworks deployed on devices and networks should be context-aware and not require users to constantly change settings when such parameters change.

Currently, tools like DHCP and Zeroconf [Guttman2001] allow a measure of automated connectivity at lower levels, but usually do a proper job only through prearranged configuration scripts and when devices and networks *know* each other, unless neither has a security policy. Without significant advances in AI, it will not be possible to let devices make 100% of the

decisions by themselves. There will be situations when user feedback will be required. Interfaces provided to users must be intuitive, and fairly high-level feedback will be expected from them; i.e., users should make high-level policy decisions about objects and relationships they can understand, rather than low-level decisions about network addresses or firewalling of certain ports.

To summarize, spontaneous interoperation of devices and networks is difficult because of the huge range of the following characteristics:

- Heterogeneity of devices and communication features
- Differences in the kinds of resources (and services) possessed and offered
- Differences in capabilities possessed for interacting with external entities
- Contexts and context-sensitive constraints that cannot be anticipated in advance
- Diversity of security and privacy policies, and trust relationships

Usability requires that computers interact with each other autonomously and present suitable user interfaces, though the latter is not a focus of this research.

Interoperation broadly consists of two processes: discovery of services and access to those services. Current systems and prior research have provided separate solutions in each of these areas. In open systems and those offering ubiquitous services, we will see the functions of discovery and access merge more and more, and existing solutions won't work. This is because the *resources* possessed could include private information that domains don't necessarily want to advertise to the wrong parties.

We can identify key hard problems for which existing research does not provide adequate solutions. One problem occurs because of heterogeneity; how do devices match interacting device requirements to local resources while maintaining security and resource management policies. The second problem addresses the need for more flexibility in interactions. Current systems adopt an all-or-nothing approach, with rigid policies that are targeted toward very specific requirements or goals. Most often, this presents a device with situations where it needs to expose more private information and allow access to more resources than it needs or wants. Fallback or alternative agreements can be reached by lowering requirements or determining exactly how much security can be risked or how much privacy can be given up. Most systems leave it to users to make these decisions upon failure. This not only detracts from usability, but also circumvents a problem for which an automated solution can be worked out. Moreover, rigid policies or tailor-made applications for domain-specific needs cannot work in an environment where context changes dynamically, and all situations cannot be anticipated beforehand. Unanticipated results and security holes will invariably occur. The solution is to use flexible and domain-dependent policies that can be easily specified in a high-level language, and can be easily modified online. A middleware for policy management is therefore essential for ubicomp.

This prospectus will show how mutually unknown devices can interoperate by *negotiating* with each other based on their policies, rather than in a manual fashion or by using static agreements. I propose to design and build such a negotiation framework that will ensure maximum possible resource access with minimum compromise of security and privacy constraints by using strong trust models and relative utility measures of system requirements. This work will assume that there is compatibility among the interoperating devices below the application layer; i.e., secure networking and transport protocols exist that are common to the entities. Given the universal adoption of MAC protocols like 802.11 and Bluetooth, network protocols like TCP/IP, and even other protocols like HTTP and SSL, this is a safe assumption. Entities also need to have some common understanding of objects at the application layer, and this is being achieved through the Semantic Web effort. I will leverage the results of this research to the fullest, with some liberties taken, inasmuch as this is ongoing work and not yet a standard; in fact, my research could well contribute to this effort. This research will be undertaken as a part

of the Panoply ubiquitous computing project [Eustice2003a] which is based on a *spheres of influence* model.

This prospectus is organized as follows. In Section 2, I describe typical kinds of interactions that occur across device and domain boundaries, typical constraints, and why a common policy framework provides the most flexibility and autonomy. In Section 3, I describe the negotiation model, its use in practical scenarios, and an analysis of the benefits. Section 4 discusses the various research issues involved in designing and building such a negotiation framework. Section 5 describes the design approach, and the dissertation plan and schedule are outlined in Section 6. Section 7 describes related and complementary research, and Section 8 concludes the prospectus.

2. Cross-Domain Interactions in Ubiquitous Computing Systems

It is fairly obvious, given the nature of ubiquitous computing and the scale at which it is envisioned, that centralized control is impossible and undesirable. Control and management of computing resources must and will be decentralized and domain-centric. In this document, any computing element or group that can act autonomously is referred to as a *domain*. A single device that exhibits autonomous behavior is a domain, as is a group of devices and resources (such as printers, speakers and displays) connected by a local network; a domain containing multiple devices interacts with external domains as a single virtual device. A domain has a single authority that imposes a common policy on every element within. Examples include enterprise networks in offices, department and lab networks within a university, and device clusters. A domain need not be defined by locality, or circumscribed by a physical boundary. Logical groups of devices that share certain attributes, are part of a social network or organization, and are governed by common authorities, are also domains, though the individual components may not always be on the same network. Organizations like ACM, or a network of friends, are examples of such domains, where the individual components share knowledge, permissions and policy. Building security and trust relationships among all domains and making them obey similar guidelines and policies is impractical for the same reason that centralized management is. Still, ad hoc associations must be supported and users carrying their personal devices need to access computing resources and be able to network wherever the presence of such infrastructure makes it possible.

Interactions take place commonly between mobile user devices and the network(s) they try to join, and sometimes also directly between devices for sharing data and sending messages (explicitly triggered by users). Examples of these are described in Section 3.1.2. Different administrative domains may also require making agreements with each other so that member devices can communicate across domain boundaries. For example, ACM might make an agreement with UCLA to allow UCLA terminals to be used to read ACM conference papers; any change in the agreement will trigger a change in the relationship and constrain the ways in which UCLA students can access ACM resources. This could potentially be done using static policy and standard mechanisms (which then cannot be ported to other scenarios), but situations that warrant dynamic formation or changes of agreements will arise. The Panoply project, (described later) of which this research is a part, deals with such issues.

Modeling our world as consisting of domains with hard boundaries, being containers for resources and contextual (or state) information, and having sets of policy rules helps us understand and provide solutions for spontaneous interoperation in ubicomp [Kindberg2002]. An identical solution can then be used for interaction between two devices (the simplest case) and between domains containing device clusters and other resources. The common theme underlying

all these seemingly different types of interactions is that they all involve service discovery and service (or resource) access. The domain-dependent variable is policy, which every domain is free to set as it chooses, and which guides the decision making of the infrastructure during interactions and also local resource management. As an extreme case, a completely open system without any security or privacy issues would allow interactions based on a *null* policy. This is a more tractable and scalable solution compared to an application-oriented approach, where an application running in a domain tries to manage policy in isolation from other applications, or by creating new mechanisms for every new type of interaction we need to support, simply because of the huge number of combinations of possible policies and contexts.

2.1. The Case for Policy-Based Management

Most domains, or environments, including single devices, will have certain common features such as networking, display, and audio capabilities. In fact, given a basic networking and processing capability, all resources could, in theory, be obtained and accessed across domain boundaries. Most devices that mobile users carry around will perform a small number of specialized functions through off-the-shelf components. There are mechanisms available to do almost anything that is computationally tractable; the number of ways to make use of resources and system capabilities is huge and is increasing. Ubicomp interaction will be based on the principle of being able to find and use such mechanisms wherever available, because user devices will have neither the hardware capabilities to perform every conceivable task nor the storage capacity necessary for all the services and data that their owners will find useful. Therefore, given that any required mechanism can be discovered and used, the problem reduces to the question of how those mechanisms can be used and offered, looking primarily from a trust and system integrity perspective. An infrastructure that must deal with domain-specific requirements while targeting interaction from a global perspective must necessarily decide the *policy vs. mechanism* argument in favor of the former. Such an infrastructure must also be structurally a middleware that lies between the operating system layer and the application layer, being independent of both.

Policy is essentially an abstraction, or a set of rules that constrain how a system can behave and how it ought to behave. This leads to the core of my thesis, which can be outlined in two parts:

i) All devices and domains have established local policies that constrains the way they can use and export their services.
ii) Interoperation among devices and domains in the absence of a preestablished trust relationship or a common set of service (or application) level protocols can be achieved through a generic negotiation protocol as long as the participants have a common understanding of resource semantics.

Policy is a set of factual and behavioral specifications that are binding on every computing element and resource within a domain. Policy must specify entities (as represented by computing devices) and their attributes, security and privacy constraints, trust relationships, security credentials, network types, resources and protocols, cryptography-based objects and protocols, data and content types, and contextual parameters like time and space. This list, which is fairly comprehensive, must be supported in a general-purpose ubicomp middleware. What looks like a huge burden for a research project to support can be handled using formal models or schemas for description and inference. Research in formal logics, databases, and semantic web tools like RDF/XML provide a head start in this area. On the other hand, users would expect different things from different policy categories – access control for resources and customization of output on displays based on user preferences and context being two disparate examples. For implementation and evaluation purposes, we will select certain categories, and will also illustrate how this can be extended to other environments for which new policies can be written without

having to reimplement the infrastructure. Some of the more general requirements for policy are the ability to deal with groups and classes of objects, specification of *general* behavior and *exception* conditions as well as meta-requirements such as setting up precedence among different policy rules.

Any system that we use to describe and reason about policy must have an ontology that defines *what* policy can specify and semantics that indicate *how* it can specify them. For example, a domain could have policy rules that describe its knowledge about objects that it is aware of (such as computer identities as IP addresses, location and time parameters, resources like storage capacity, printers and displays) and the relationships (such as access rights) among such objects. To enable interoperation, different domains must have a common global ontology for policy as well as a common syntax for a high-level description of resources of shared interest. There will, of course, be domain-specific data and objects that do not need to be described using the same policy language, but the language should support such description, i.e., it must be expressive. The policy framework or manager must also be able to reason with a set of policy rules and provide some guarantee of correct results when action decisions are made on the basis of local policy. The presence of such a framework precludes the need to invent specialized protocols and mechanisms for diverse security and resource access requirements.

Two important arguments are necessary here.

i) Different domains need to agree on some issues in order to interoperate, unless their administrators and users wish otherwise. The bare minimum that must necessarily be common, while leaving individual users the maximum independence, is a shared policy description framework, a way of describing common resources, and a communication protocol for transaction based on those policies. My research will remain useful even if some resource types, protocols or policies may become standardized because of wide usage, at a later date.

ii) A domain administrator with sufficient programming skill could write up an application that implements the collective system policy. There are several significant advantages to using a policy language approach. This approach is obviously more flexible. Both online (i.e., while the system is running) and offline changes, and the addition and removal of policies are possible; in contrast, an application will have to be examined, modified, recompiled and redeployed. A number of policies will be specified by naïve users; neither will such users be able to modify system applications nor can designers anticipate all possible preferences that users might have, especially since such preferences may have a cross-application impact on the system. A policy manager also allows a modular approach, so that resources can be added or removed easily, and the manager takes care of resolving newly added policies with existing ones. Policies must therefore be easy to write at a high level, and can be ambiguous in certain ways, notably related to context. It is the policy management middleware's task to clear ambiguities and make specific decisions in various contexts.

There is nothing novel about using policy to control system behavior. Most systems use policy for flexibility and extensibility and also to impose constraints on the usage of system resources. While traditional uses of policy have been domain-specific and meant for local interpretation, I propose to use policy as a tool for ubiquitous bidirectional interoperation.

2.2. Domain-specific Aspects Controlled by Policy

The basic operations through which a middleware can enable interoperation to establish dynamic relationships using local policy will be described in Section 3. Here, we take a look at three key high-level functionalities that form a fairly complete combination of issues that are impacted by inter-domain interactions, and which therefore must be specified in system policy:

resource management, security and access control, and context-awareness. Policy will also specify deontic concepts like obligations and permissions [Kagal2003a], and meta-constraints for priorities and for resolving modality conflicts.

2.2.1. Resource Management

Every domain possesses resources ranging from high-level ones like printers, displays, audio devices, sensors, actuators, and network connections to more basic resources like amount of bandwidth, disk space, files, data, and even memory blocks. It must export interfaces for such resources to users and applications. A single device manages resources through an operating system, and a network or cluster can manage its resources in many ways, ranging from a distributed operating system with tightly coupled devices to a loose federation of devices that share a single server or gateway to the outside world. Policy can be used to describe and constrain the way each of these resources can be used, and to perform resource allocation when multiple clients or applications have similar requests. It can describe how the usage of one resource is dependent on (or constrained by) another. High-level resources usually are dependent on lower-level resources, and any actions that are requested by clients could impact the behavior of multiple resources at different levels. Also, requests for resources at a high level, common when neither the requester nor the owner have knowledge about each other's possessions, could be translated into requests for access to lower-level resources at the owner's end. For example, Bob carries a PDA with him to a coffee shop expecting to get internet connectivity and be able to use a particular network protocol through the shop network's gateway. The shop network's policy for connectivity and protocols impacts low-level resources like network bandwidth and the amount of buffer space it is has available. Therefore, what is a simple requirement from the PDA's point of view involves a more complex interplay of policy rules governing resources at the network's end. In a highly dynamic environment, with the number and nature of clients in constant flux, a policy-based framework would be necessary to enable interoperation and to make sure that desired system behavior is exhibited. Such a system could be used to monitor conflicts and either solve them using meta-policies, or report them through appropriate interfaces. Systems like Keynote [Blaze1999] do such conflict checking, though in a static manner and using a restricted policy language. In my research, I am interested mainly in variable high-level policies (that users can set up) and their interplay with more static low-level policies in different contexts.

2.2.2. Security and Access Control

Most systems have some kind of policy for security and access control, based on a local measure of trust and an idea of what it could take to compromise a system and misuse its resources. Security policies include filtering remote service access based on identity and port (this is done using firewalls), and memory and file access restrictions to prevent buffer overflow attacks and mitigate the threat of viruses. Access control and privacy policy rules are used to answer questions like "*who is allowed to access a particular resource?*" and "*what kind of authentication is necessary?*" In ubiquitous computing, *trust* will play a huge part when mutually unknown computing entities interact with one another. Access control policies will be specified using a local idea of trust, since it is impossible to establish trust relationships between every possible pair of devices in the world. As with resource management, it should be possible to describe security and access control rules at a high level, with the policy manager deciding what mechanisms to use in an ad hoc manner. This approach is highly flexible compared to many existing security frameworks that would require re-engineering when faced with new requirements. Different resources may have different access policies, and it is impossible to anticipate all permutations and side effects of these beforehand. Also, higher-level security

policies (like setting security levels in the Internet Explorer browser) could impact behavior at a lower level (for example, which ports must be opened). A user or a system administrator should be able to declare policy and leave it to the system to resolve conflicts and ensure no violations during interactions with external devices (how this can be done will be shown later). Meta-policies, such as those pertaining to security/privacy conflicts, could help a policy manager with decision making. For a system in which new modules could be added dynamically, policy can be used to monitor the security impacts and prevent violations. Often, it might be important to have local policy rules kept private, since the exposure of these may inadvertently release private and sensitive information. This could enable malicious entities to discover security holes or make use of the nature of the policy itself in order to mount attacks.

2.2.3. Context-Awareness

Context-aware computing makes computers more intelligent from the point of view of a user, since they provide customized service to users based on the *context* a user finds himself in. In a ubiquitous computing world, the same application or resource provider would behave differently based on its perceived context. A context-specific application of a general policy would resolve to a set of lower-level policies. Learning a user's behavior and anticipating his needs is an artificial intelligence problem, but there are systems issues in determining exactly what the context is and what is of relevance to the current context. Here, context can be used, not as a policy category by itself, but as an added dimension to resource management and security policies. Location and time are the most common and widely used contextual parameters. For example, if I am in my car and need to find a gas station, I would like the application on my PDA to find the closest gas stations, rather than give me a complete list or prompt me to specify my location. Here the policy simply states that a gas station be located when gas runs out; the device has some way of sensing gas level, or could get such information explicitly from the user through some interface. I might like my home TV to change maximum volume levels gradually based on time, or prevent R-rated content from being screened during certain hours. Other types of contextual parameters could be considered, like applications behaving differently on my PDA based on whether I am in a public bus or in my car. Of course, for a truly intelligent environment, any system would require sensors that provide context information; various sensors are in use today, and innovative ways of using them is the focus of a lot of ongoing research. Lastly, any policy framework that supports context awareness must allow generalizations over objects and context to be specified as well as exceptions, which can be used by systems to make quick and appropriate decisions. In practice, it is usually not necessary to *completely* specify context-adaptive behavior, which may be an impossible problem; partial specification and probabilistic reasoning could be used for decision making. The policy management system should be able to extrapolate policies to a context where direct rule lookup is not possible or cannot be inferred.

No policy framework has rules for resources, security and context awareness written in complete isolation from each other. The interplay of these different functions can get very complex in a dynamic and heterogeneous environment.

In this section we have seen why local policies for devices and domains hold the key toward achieving a global scale decentralized framework for interoperation. In the following section I will describe how policy can be used to achieve this vision, what kinds of operations are involved in this process, and outline some practical scenarios.

3. Research: Ubiquitous Interaction through Policy-Controlled Negotiation

Consider the following example of an inter-domain interaction. When Bob enters a Starbucks store, his PDA (or laptop) must resolve its policy with the Starbucks (a domain represented by a WLAN) policy in order to obtain resources, in return for which it might need to hand over some private data. Bob's device will have to abide by Starbucks' policy as long as it is a part of the local network. The resources required by Bob's device and the private information that the network demands are not rigidly enforced by a standard. Bob still desires his device to join the network and obtain some service, and Starbucks desires to have Bob as a customer. This requires some adjustment of demands on the part of Bob's PDA as well as on the network manager, without one party completely capitulating to the other's rigid policy. A second example of interaction is the addition of a new DVR to a home entertainment network, connecting it automatically to a TV, speaker and a home computer and having it behave in accordance with pre-set network policy and owner preferences. Or, consider how the computer in Bob's car can receive sports and weather updates constantly from different access points offering ubiquitous networking services for a price.

To support the interactions in the above examples, which mainly involve service discovery and resource (plus data) access, the participating entities (or domains) must come to a working agreement. The policy management middleware on interacting domains must *negotiate* to reach such an agreement, while maintaining local policy in dynamic conditions.

3.1. Policy-Based Negotiation

Negotiation is a policy-guided operation through which devices can make requests of each other and decide whether or not to give access to local data and resources. In the most general case, each participant's local policies are private and unknown to the other, and a number of these might conflict. Each entity starts off with certain requirements, or targets, or goals, in mind; negotiation guides them to a point of agreement or *compromise* through the use of suitable meta-policies and heuristics. It is, in effect, a process of policy resolution and conflict management, except that each entity has partial knowledge of the other's policy, state and goals. Keeping policy private is not a random assumption; exposing policies, especially access control rules, could have serious security implications. A malicious agent may potentially take advantage of such knowledge to compromise systems; this has been demonstrated in widely used network security protocols. Policy resolution through negotiation with full knowledge is also a non-trivial operation, though it might lead a more satisfactory agreement for both participants.

The negotiation that I am focusing my research on is different from various other protocols that are used for interoperation. This is a bi-directional protocol, without any side being constrained to be a client or a server. It also does not follow a script (in other words, a shared policy) whereby only state matters, and not local policy. Ubicomp negotiation is dynamic, with local state changing during negotiation; this in turn necessitates a reevaluation of the low-level policies that constrain service exposure and access permissions, and the goal one expects to reach at the end of the protocol. Developing good evaluation functions for such goal evaluation is an important part of negotiation, and this could also involve meta-issues like the time taken and the level of trust gained in the opposite party.

A negotiation protocol resulting from this research will not be targeted toward a specific domain. Two domains can negotiate as long as they share a common semantic framework for the description of their resources and policies. An infrastructure like the Semantic Web and an expressive policy language are sufficient for such a negotiation scheme to be applicable

ubiquitously. For example, negotiation is going to be difficult, if not impossible between entities that have a totally different idea of what the resource "*display*" signifies. (Note: this does not preclude security domains from using diverse strategies and heuristics, which need not be understandable or familiar to other domains.)

Ubicomp interactions typically involve wireless communication between mobile devices and network access points, though these aren't the only types of interactions. Software agents that are not device-based could also interact with each other through web links. Client-server transactions could be performed on behalf of users. Policy-guided negotiation protocols are as applicable to such transactions as they are to ad hoc wireless associations, since these cases also involve private data and policies. Good research has been done in these areas already, such as the P3P standard [P3P] and automated trust negotiation [Winslett2003].

3.1.1. Negotiation Model

In this section I describe a very general model for policy-based device negotiation. This model considers two-party negotiation, though it can be extended by induction to an *n*-party case.

Given that there are two computing entities C_1 and C_2, which could be devices or networks that can operate autonomously:

- C_1 has a set of resources R_1, a policy P_1, and a set of services S_1. R_1 is general enough to incorporate high-level services like printing or display, and also low-level items like individual data items, memory space, and even the policy P_1, being local content. The policy P_1 is general enough to describe both the state of a device or a network, values of relevant context parameters like time and location, operating rules that describe access control of resources and data objects, resource allocation, use of security mechanisms, context-sensitive behavior, and deontic concepts such as obligations. If C_1 is a network, P_1 would include information about resources allocated to or accessible to individual computers within the network. S_1 broadly describes a set of applications or services that serve a local need at C_1. All resource requirements in a given context can be derived from S_1; if not available locally, they can be obtained from the external environment.
- Likewise, C_2 has a set of resources R_2, a policy P_2 and a set of services S_2.
- It cannot be assumed that C_2 has knowledge of R_1, P_1 or S_1; likewise C_1 might not know R_2, P_2 or S_2

The result of C_1 negotiating with C_2 is a relationship that allows C_1 access to a set of resources $Q_1 \subseteq R_2$, and C_2 access to a set of resources $Q_2 \subseteq R_1$. Here, Q_1 and Q_2 are the maximal sets of resources obtainable within the constraints imposed by P_1 at C_1 and P_2 at C_2, and the partial knowledge possessed by either entity. Q_1 and Q_2 are subsets of the requirements derived from the need to make S_1 and S_2 function. An ideal negotiation result would occur if Q_1 and Q_2 equal the set of requirements before the first message, but that may or may not be possible in the absence of an oracle that has knowledge of both sets of R, P and S.

At a conceptual level, negotiation is a tradeoff, a process of give-and-take. Two types of resource compromise are possible. If a resource can be measured quantitatively, the participants will negotiate to a point where the quantity of resource obtained is less than or equal to the original desired level; if the resource can either be given or not given (in terms of access permission), the final result might allow access to an alternative resource that might serve the purpose but is less *valuable* or less *desirable* according to the consumer's policy. Obtaining access to a resource might involve some *sacrifice* on the part of this consumer, or some behavioral *obligations* could be imposed upon it.

The negotiation protocol is heuristic-driven, and will in all probability be a *best effort* solution. Heuristic functions could be computed from meta-policies that resolve conflicts, models

that evaluate trust in the opposite party at every messaging step, *utility* functions that help to re-evaluate the goals, or even extraneous factors like efficiency (expected time to reach an agreement). A negotiation protocol, irrespective of the scenario it is used in, has a security and trust model at its core, since the result of negotiation is the assignment of access permissions. Such a trust model guides the gradual exposure of resources and policies that are initially hidden.

The actual negotiation protocol revolves around a very simple and generic state machine, with a very limited set of message types, such as requests, offers, policies and queries, and these will be described in detail in Section 5. It terminates when either party decides that further negotiation is fruitless or undesirable.

3.1.2. Example Scenarios

Scenario 1

Bob walks over to the local Starbucks coffee shop with his laptop and PDA, each of which runs a policy manager capable of negotiation. Starbucks runs a wireless network managed by a policy manager that runs on the same machine that controls the access point(s). The wireless network's policy is used to let the network decide how and when to let client devices join, and also governs network interaction among devices in the network, and services used by these devices through a wireless connection. The primary resource exported by the network is Internet connectivity at various levels of bandwidth. The PDA by itself has no useful service to provide to the network, and is therefore very much in the position of a client (or a supplicant). Still, Starbucks would like Bob to be a customer, and therefore it would be in the network's interest to reach an agreement with Bob's device and provide service.

The PDA is running certain applications that require network connectivity, and therefore it initiates negotiation by asking the network policy manager for permission to join the network and obtain network connectivity at a certain bandwidth. The network's policy is to grant a minimum network bandwidth to all members, while extra bandwidth would require some compromises on the part of the clients. In this case, the Starbucks policy manager offers the minimum bandwidth to Bob's PDA in response to its request. But the PDA is running an application that allows Bob to listen to streaming music from his home computer. The PDA can proactively cache up to 5 minutes of music, but needs to reconnect soon, and also requires a higher bandwidth than what has been offered. So the PDA requests a higher bandwidth connection. The network, in return, and in accordance with its policy, asks for Bob's email address along with a policy statement which indicates that it will send Bob occasional emails about promotional offers, and also offers to make Bob a *preferred member*, which will allow Bob to obtain discounts at various outlets. The PDA evaluates the request for Bob's email address based on the offers. It then poses a counter request for a credential which indicates that Starbucks does not have a spam policy and also imposes an obligation that Starbucks will not send more than one email a week. The network finds that these requirements do not conflict with its policies; it sends a certification from the *Anti-Spammers Bureau*, counter-signed by Verisign. The PDA considers this information in conjunction with the real-time networking requirement imposed by the streaming application, and decides to give its email address. Along with network join permission, network usage obligations are sent to the PDA, one of which indicates that an email client can be run only if it is uptodate. The PDA is running an older version, and since it knows that Bob typically checks his email around that time of the day, it requests a patch from the network policy manager. The locally cached patch, which has no access control policy, is sent to the PDA. If the patch isn't locally available, the PDA will not be able to run the email client; if it violates that policy (as can be detected by a network port scanner), its membership within the network might be revoked using a firewall (unless the PDA obtains the patch directly from the Internet).

When Bob starts his laptop, its preferred policy is to connect to the PDA if that is already on and can function as an Internet gateway. In this case, it can do that, and therefore the laptop does not need to negotiate with the Starbucks network and can connect to the PDA (this negotiation is very minimal, since the laptop and PDA have, presumably, a prior established trust relationship). Lastly, as a bonus for becoming a preferred member, when Bob goes to the counter to buy coffee, he gets a discount.

On a PDA with a rigidly defined access control policy of "*default deny for all private identity information*," when the question of giving email address information comes up during negotiation, a pop-up might be flashed to let Bob make the decision.

This scenario basically illustrates how users can obtain their services continuously using mobile devices and the availability of ubiquitous networks, without the need to establish a prior trust relationship with each of these. Today, a Starbucks network is completely open for any device to join, with devices being able to do basic Internet browsing. The network makes no security guarantees nor does it provide any other services. This is because Starbucks uses free access as a way of attracting people to its stores. In the future, stores like these and others could discover the potential of providing more and varied services, not all free, and in return users could become more and more wary of possible privacy violations. When scenarios get even a little more complicated than those where there is either no policy or a very rigid policy, negotiation is the only proper way for systems to interoperate. Then, each side needs to release only the minimum private information necessary to establish a network connection and permission to access resources. Resource access can be granted and gained at varying levels, and identity-based relationships do not have to be preestablished. Here, the only place where an identity association is made is when the network produces an Anti-Spammers Bureau certificate.

This scenario is not far-fetched, and indeed may play out at most wireless hot spots in the fairly near future. Google has been trying to acquire Wi-Fi networks in order to provide free wireless Internet access everywhere for users carrying mobile devices [Google2005]. Access points will typically be set up at local shops like Starbucks, malls and other places where people concentrate. The model is very simple right now; users get free Internet access and in return Google obtains information about user browsing patterns through its control of the gateways, and as most people use its search engine. The satisfaction with this state of affairs may not last, especially if Google gets more inventive and is able to obtain private user information and possibly send targeted advertisements to device browsers. Google may also collaborate with web service providers and provide different kinds and grades of service to users. Users will begin to be more concerned about their device privacy and security. With all kinds of possibilities opening up, there is a huge potential for automated negotiation of the kind described above in the Starbucks example.

Scenario 2

There are various scenarios that exhibit similar dynamics to the one described above. One such scenario, which sounds quite futuristic as it involves more advances in AI and security infrastructures than we currently possess, comprises a mall where multiple shops have individual wireless networks, each of which advertises its offers and prices. Bob's E-Watch, as part of his personal device network, knows what items he needs or wants, the value he puts on those items and his priorities. Based on offers received from various shops in the mall, the device could negotiate prices and discounts and potentially carry out transactions (which would require giving up credit card information) that serves Bob's best interests in terms of utility and profit (of course, every shop computer will be trying to do the same). The policy management and negotiation framework that is proposed in this document will enable these kinds of transactions.

Scenario 3

This scenario primarily illustrates a seamless system configuration without manual intervention, but also indicates bi-directional negotiation and relationship formation.

Bob buys a network-enabled DVR and plugs it into the home network. The DVR has a default policy (at manufacture time) that requires it to search for display devices for output and cable services as input. Currently this is done using wires, but we can expect DVRs and televisions to communicate wirelessly in the future. When the DVR is added to the network, it makes known its requests, and the ensuing negotiation will require that the network policy manager be able to examine the DVR to ensure that it has an updated version of the software and inform the DVR of Bob's policy of not allowing output of R-rated content during the daytime. Since the DVR is brand new, it must accept all policies from the network and agree to abide by them. Failure to do so will be construed as an indication that there is something wrong with the DVR (i.e., it may have been compromised). Bob is also very concerned about media piracy, and will only allow his closest friends and family access to content stored on the DVR. Therefore, when some of his co-workers come to his house, and each of their personal devices negotiates to join the network, all the devices will be allowed connectivity and the freedom to surf the net and check email. However, a device carried by his close friend Frank will be given access to the DVR and will be able to access the stored episodes of "*Friends.*" This device, through its policy settings and knowledge of the current context, can infer that Frank wants that particular content and this content should be searched for in this location.

Scenario 4

Bob goes to a ubicomp conference where he meets Frank, and both discover that they share a common research interest and could potentially collaborate and share information about papers, conferences, job openings and so on. Bob is already a member of a group of researchers who collaborate on such topics and can share information. Since the information that they want to share is stored on their devices or accessible through their personal devices, Bob and Frank augment policy on their respective handhelds so that each is allowed to access ubicomp-related and other private information. The devices negotiate with each other (explicitly triggered by their respective owners) and exchange ubicomp research related information, email addresses and instant messenger IDs. Here local security policies are very valuable because they prevent a device from handing off any information that could be misused by the other; a trivial example is that of not providing access to papers that are copyrighted and therefore cannot be shared. Bob is also a member of a worldwide research group of collaborators who can share information of the type that Bob and Frank want to share. Group interaction obeys certain policies, such as no spam. Bob's device introduces Frank into the group by granting a permission that Bob's device is delegated to provide, and is allowed to do so by group policy.

A number of complex interactions take place during this negotiation, and a part of the solution rests in the Panoply sphere model, which will be described shortly. Apart from that and some data organization and searching techniques, the negotiation model that was described earlier is sufficient to realize this scenario.

3.2. Benefits/Advancement of State-of-the-Art

This research is one step ahead of prior work in the field of ubiquitous computing, notably the intelligent spaces augmented by sensors and actuators. We take a top-down approach by looking at *ubiquitous interoperation in its entirety* and at the state of current technology, and then making justifiable assumptions about expected advances in the near future. As far as we know, no framework currently exists that allows ubiquitous ad hoc negotiation and resource sharing among devices with *no prior trust relationship*. Such a framework will be sorely needed

in the near future, and can be used in any large scale environment with multiple administrative domains. One criticism of this approach could be that it adds something more that devices need to agree on, or have deployed, before they can interact. But as explained earlier, the set of functions identified as being part of a policy negotiation framework is probably the bare minimum needed for any kind of ubiquitous interaction.

Looking from one point of view, a *general purpose negotiation framework* is a part of the semantic web effort, and complementary in some ways. Domains don't need to share common and static policy, context, resource sets and applications in order to negotiate. This enables semantic bridging at the application layer, which is essential for interoperation. The actual negotiation process goes further than anything attempted before, being unscripted, dynamic and context-sensitive. Other protocols that have proved very useful, such as DHCP, perform specific tasks in specific contexts and have rigid semantics, making it very difficult for an average user to tweak behavior. Still other protocols have emerged with newer technologies, and which are more flexible in one way or another:

- Network protocols that attempt to guarantee QoS.
- Grid computing [Czajkowski2002] service negotiation protocols, which are flexible if the services are known in advance, even though these are not portable to a mobile computing scenario.
- Automated trust negotiation [Winslett2003], which enables flexibility in exposure of private information on the web, though using a static policy as the basis.

My research target is to build a solution that will work with dynamic context (and consequently, policy) changes, and where the final result can be negotiated, and is a set of resources, rather than a single resource, so that the intermediate and final goals can be reevaluated. For example, let us say device D is negotiating with network N, when suddenly device F enters the network (with which it has a prior established relationship) and exercises its access control rights. The resource availability and access possibilities change enough for N to have to modify its negotiation strategy with D.

This research will also benefit the field of computer security by providing *context-sensitive access control*. Most access control systems are conceived in static contexts and are not usually scalable. Still, models for open systems like RBAC, delegation, and webs of trust have established concepts that are very valuable in preserving security and privacy. Since negotiation proceeds from the assumption that all resources and preferences are private and the exposure of such resources or granting access permissions can be done only if policy allows, *security is automatically provided at the core*. Using a trust model as a guide to negotiation, with trust levels than can be enhanced or lowered based on negotiation message content, is one of the tasks I plan to perform. "*Prevention is better than cure*" should be the motto for any system administrator who is concerned about security. If mechanisms like QED [Eustice2003b], developed in our lab, are used in conjunction with a security-conscious negotiation protocol, users and administrators can proactively set security constraints under which interaction is allowed, rather than using tools simply to detect and patch breaches after they have occurred. The security basis of interoperation is not based purely on identity, but on characteristics and policies of the entities. Thus the *level* of interaction that each negotiating party desires could be fine-tuned using security policy as the basis.

A useful side benefit of this project could be static verification of security and privacy levels of a system. One could model systems with a set of standard resources and access control policies and run experiments (possibly by simulation) to make security assessments. Certain actions or policy rules that compromise security could potentially be discovered. The probability of an activity leading to a situation where a malicious entity could mount a security exploit could be inferred. System users could then decide whether or not the utility provided by the task outweighs the risk, given that there is no suitable alternative.

The field of ubiquitous computing is a mix of system and AI research, since it promotes a *non-intrusive paradigm* where computers can proactively and intelligently perform tasks that users want done. Policy management and negotiation enables users and domain administrators to set policy and then let devices and agents interoperate on their behalf with a high confidence of correct results, thereby remaining faithful to this paradigm. My research will also investigate how principles of game theory and utility theory can be applied to device interaction. There are distinct parallels between negotiation strategies and the game theory models of adversarial and cooperative paradigms, and the concept of relative benefits (primarily security risks and trust) in utility theory research could be applied to policy resolution. My research will therefore benefit the community in general by applying such AI and theoretical concepts to real-world systems and observing the results. It also advances the area of autonomic computing and makes computing devices more *usable* by ordinary users, who don't have to worry about the internals and low-level behavior of their computers.

This research will be conducted within the framework of the Panoply ubiquitous computing system [Eustice2003a]. Panoply provides a scalable, secure framework by building on the *spheres of influence* [Eustice2003a] paradigm, which divides the world of computing entities into spheres, which could be physical domains, social groups, personal clusters and even individual devices, each sphere having a security boundary and being able to scope policy and context. Negotiation is a tool to enable both inter- and intra-sphere interaction. Negotiation concepts added to Panoply and spheres result in a powerful model for the structuring of device groups and establishment of device relationships.

4. System Research Issues

The core of my research deals with traditional *distributed systems* issues, namely the negotiation protocol, security, fault tolerance, and scalability (with respect to the sizes of the policy and resource sets). *Programming language* issues like expressiveness and semantics are involved in designing languages to express policy and describe resources. Managing knowledge bases that contain state information and policy rules are issues that have been long handled in *artificial intelligence*; functions like indexing, retrieval and reasoning fall within this area. The *intelligence* involved in autonomous negotiation, which includes the heuristics, cost/risk/benefit tradeoffs and negotiation goal reevaluation, when applied to service discovery and access control, involves a subtle mix of systems and AI research. Likewise, theoretical evaluation falls under the AI domain, whereas performance and functionality evaluation falls under the systems domain. An assumption in my research is that policies are specified by users and system administrators. Learning policy based on user behavior is outside the scope, though it is an important ubicomp topic that is being pursued by others. The interplay of policy with specific contexts provides a rich enough space for research without needing to add AI learning techniques. With this caveat in mind, we look at individual research issues in more detail.

(In every subsection, the key research contribution is either highlighted in bold and italics, or is summarized at the end. Topics without highlighted text are relevant to the ubicomp interoperation problem but outside the scope of my research.)

4.1. Discovery and Configuration of Device Communities

The theory and design of *device communities* that can manage resources (in a centralized manner), impose policy, and ensure secure communication, both within and with the outside world, is an important research topic in its own right. Such communities map to the domains that

are the units of interoperation in a global scale ubiquitous computing system. Modeling device groups and handling associated research issues like device and group discovery, event management, secure networking and firewalling, which are orthogonal to the negotiation problem, are being dealt with by Kevin Eustice in a project called *spheres of influence*. The spheres concept combined with negotiation is expected to result in a middleware for ubicomp called *Panoply* [Eustice2003a]. My research does not absolutely depend on spheres, since groups can be virtualized by a single computer, and existing security and MAC protocols could be used for wireless communication. Still, negotiation added to Panoply will demonstrate a powerful ubiquitous computing model.

4.2. Flexible and Extensible Negotiation Infrastructure

The negotiation model discussed in Section 3 is very general in nature and can be applied to a wide variety of domains, scenarios and resource classes. The negotiation protocol needs to be flexible (i.e., the message types need to be generic enough), and one such approach is described in the implementation section (Section 5). The policy languages also must be expressive enough and provide enough ways of reasoning so as to handle a number of interoperation situations we will observe in a real world. In practice, we will be experimenting with a few selected examples, and future research will reveal exactly what classes of scenarios are representative of the complete range. And of course, the target of any systems project is to be extensible. ***Ideally, we must strike a balance so that the framework is flexible enough but not so loose that it would leave implementers and users with significant work to make their devices and networks interoperate with other domains. In the context of this research, this entails identifying a set of objects and operations that are common to and can be used in most situations.*** Of course, some exceptional situations will exist, but if these form a very small percentage of all cases, it would be better to handle them separately, rather than stretch the model and make it weaker. We have already made the commitment that the only thing a user or a network administrator needs to do is to describe their preferences and constraints in the form of policy rules, without worrying about system internals, and the middleware will take care of the rest. The tighter we can make the model, the better, but the extent of tightness can only be determined after building prototypes, creating and experimenting with application scenarios iteratively. Experimentation with real-world scenarios allows us to model their requirements and discover further commonalities among different resource types, or policy characteristics, which could then be incorporated into the infrastructure.

4.3. Policy Expression and Reasoning

Expressing and reasoning with policy is one of the most important aspects of this research, and the efficacy of which determines how successful the final result will be. ***Therefore one of the targets of this research, and the key building block in a policy manager that can negotiate, is an expressive policy language suitable for ubicomp that provides primitives for reasoning, namely to pose queries, change policy and resolve conflicts.***

4.3.1. Policy Language

There has been a lot of research in policy languages over the years, and they are, for the most part, special-purpose, tailored to solve domain-specific problems. Recently though, there have been efforts made toward building policy languages for large-scale projects like the semantic web and pervasive computing. They do consider ubicomp interoperation problems

described earlier in this document, and inspire us to ask the following questions before we proceed to choose or design a language:

- What are the things we will be talking about in ubicomp that will be represented using the policy language?
- How do we represent those things in our language (*syntax*)?
- What do the things represented in the language mean, and what do the statements (composition of different things) constructed in the language mean (*semantics*)?

Policy languages are not, and don't need to be, general-purpose programming languages. Their requirements closely match knowledge-based systems, which is a well-researched subject in AI and databases. The key uses of such a language are to represent knowledge about a system and its behavior. Ontology for the web and for pervasive computing is a subject of ongoing research – a few of the results being RDF, DAML+OIL, OWL and SOUPA [Chen2004]. SOUPA specifically targets the semantic web and ubicomp. These frameworks allow expression of contextual parameters like time and space, persons and devices, events and actions, and behavioral constraints. My research will leverage this work, and will also investigate whether extra features are needed for ubicomp in general and negotiation in particular.

Syntax is not really a research topic, but is important to the extent that it provides a common language that can be understood on a cross-system and a cross-application basis. The *semantic web* is a common framework for sharing data across systems and applications. The RDF language and the XML markup language are frameworks that are widely used and are rapidly developing into standards that everyone uses to communicate information and data. The work involved in designing languages is in determining the low-level and high-level constructs; the latter will be used by policy writers to write policies. Seamons and others [Seamons2002] outline various other high-level properties that are necessary for a policy language that supports trust negotiation, which is a special case of our more general model, though some of their suggestions remain valid. They suggest that the language should support dealing with credentials, transitive dependencies, chains, private variables and policies, have well-developed semantics and be monotonic. The last requirement is not essential for general resource negotiation, since it puts restrictions on the functions that the protocol can perform.

The most important part of the policy language is the semantics and the *reasoning framework* that allows some meaning to be extracted from a set of policies. Such reasoning also allows us to query a knowledge base containing policies to obtain state and behavior information; it allows us to check for conflicts, add rules and so on. Such reasoning must have some formal logic backing it so that we can get some guarantees that a policy manager or *negotiator* gets an answer that is correct and is consistent with the knowledge base. Properties such as soundness and completeness could be proved formally. Research is going on in the selection of a formal logic for ubicomp services, with some researchers claiming deontic logic to be the most suitable [Kagal2003a]. This research could uncover further clues in this respect. Let us consider first order logic, which allows one to describe objects and relationships as well as collections, and has a sound (and complete with some caveats) reasoning framework. The interesting question is whether it is too general; for example, if we were reasoning with time, we could use temporal logic, a subset of first-order logic. Deontic and modal logic constructs like obligations and permissions are also special first-order logics. Whether or not higher order reasoning techniques, like meta-reasoning with relationships, are required is a research question. As a starting point, first-order logic seems adequate for the purposes of this research. In fact, logic programming languages, which have strictly less expressive power than full first-order logic, may also suffice for my research. Further progress in this work will prove or disprove the suitability of first-order logic and logic programming languages for ubicomp policy.

Managing knowledge bases containing policy also poses interesting research challenges in systems and logic. Indexing and retrieval must be efficient. The policy framework must allow different techniques of reasoning, like forward and backward chaining, for adding rules and

posing queries respectively. Truth maintenance is another technique whereby the consistency of the statements in the knowledge base is maintained across state changes. These are well-researched techniques in AI literature, and my target is to investigate whether they can be applied directly to ubicomp policy management, or whether they need to be adapted in certain ways.

A potential area of research to comment on in my dissertation would be to investigate how entities that use different reasoning frameworks and languages could negotiate with each other, where they only require a common understanding of negotiation primitives and the objects and relationships they are dealing with.

4.4. Security

There are two aspects of security that we need to consider during the course of this research: i) the security benefits that a policy-guided negotiation framework provides to a system offering ubiquitous services and forming ad hoc relationships with un-trusted devices, and ii) securing the negotiation procedure itself.

We must consider the first aspect in the context of ubicomp, and ideally produce a balance between open interaction and system protection. Most situations in ubicomp that require interaction among mutually unknown devices will, in all likelihood, turn out to be safe, even though one cannot convince the other completely of its trustworthiness (by producing credentials, exposing system information to the other). I claim that having a single module managing policy for a domain and performing negotiation will not only enhance security, but also promote a new way of thinking about security. Previous ubicomp projects have started off by concentrating on the openness of their systems, to the long-term detriment of security. A policy manager has a *safety first* motto. It ensures that a particular interaction is disallowed if it is guaranteed to lead to a security breach. From there on, depending on the likelihood of system compromise, the level of interaction (or resource accesses) can be negotiated through a protocol. Note that my research concerns the proper use of available security mechanisms (based on cryptography or otherwise) in a ubiquitous computing environment, particularly in negotiation scenarios, and in the process, promote a security paradigm. Standard mechanisms such as virus scanners, intrusion detection techniques and systems like QED [Eustice2003b] could be used to assess security risk.

One reason why fairly complex systems invariably possess security loopholes is that they possess various different modules that perform particular tasks or handle certain types of applications, each of whose security requirements interact in unpredictable ways (and some of which may have misconfigured security). A policy management middleware will be able to handle multiple modules, each having security policies specified independently. High-level policies could also impact lower-level policies (dealing with low-level resource constraints) in unpredictable and unsafe ways. For example, administrators could specify rules for how a high-level service (like a network file server) might be accessed, but there might also be rules specifying which memory blocks can be accessed, or how much buffer space it could use. Obviously, a human system designer or user cannot think of the huge combinations of possibilities that arise due to the interplay of these different policies. As all these policies are specified in the same language and have the same semantics, the policy manager can use formal reasoning semantics to determine if a particular decision, if made, would have side effects resulting in policy violations, either for other modules or at lower levels. Conflicts can be then managed using meta-policies or measures of risk-benefit tradeoffs. Research will tell whether such security benefits are evident in real-world scenarios. A lower priority research task will be to investigate whether the policy manager can contribute toward static security property verification, given a set of policies and a particular context.

The security of the protocol itself can be ensured to some extent through the use of cryptography and secure network protocols like TLS. On the other hand, malicious devices could

use certain aspects of the negotiation protocol structure to infer the private policy or the presence of guarded resources at the negotiator. For example, what is the guarantee that one party will return a suitable offer it has implicitly agreed to return through a counter-request? Whether such threats are real, and how policy managers can use suitable strategies to foil such attacks, is also an interesting research problem.

4.4.1. Trust and Access Control

Policy-guided negotiation is a flexible way of performing access control. Traditional access control models are neither flexible nor scalable enough for ubicomp. ACLs and capabilities can be used within local domains but are unsuited for ad hoc interoperation. Certificates and delegation are very useful, but can be used in limited ways and require some form of identity-based trust relationships. Role-based access control has proved to be very flexible and easy to use, but does not scale in its most basic form, since one cannot anticipate all possible roles to be assigned to all possible entities. Researchers have argued [Minsky2000] that the semantics of usage of basic AC mechanisms, such as credentials, delegated permissions and roles must be left to individual domains, and specified as policy. Also, recent research has addressed the need for ubicomp access control to be based on a very general notion of *trust*, which could be specified almost as a continuum and is not purely dependent on identity, but on actions and observations. ***My research objective is to be able to fine-tune the level of interaction that a negotiation achieves using a risk-benefit analysis, and a trust model that is necessary for the assessment of risk and design of negotiation heuristics. It is also likely that an existing trust model will be leveraged for negotiation, though it might need to be enhanced so that it satisfies the properties described below.***

The trust model needs to have certain properties, such as being independent of domains or resource classes. Different domains need not share any common notion of trust, which is based purely on the measure of importance assigned to the safety of its resources and any evidence that is given to it by a negotiating entity. It must be fairly fine-grained, so that relative trustworthiness of entities in particular contexts can be compared. This evaluation is used by the negotiation protocol to make strategic decisions. Trust could be gained through production of credentials, agreement to abide by the policy of the network being joined, willingness to share resources and willingness to expose system characteristics for examination [Eustice2003b]. In practice, this could contribute to the building of webs of trust, which can only make interactions easier and safer. Logic for trust could be developed by integrating the trust model and the policy language.

This research could also make contributions toward context-sensitive access control, a relatively recent topic that customizes access control to relevant context, which is something the negotiation protocol also strives to achieve.

4.5. Negotiation Heuristics and Strategies

One of the most interesting research directions that will be investigated in the course of this project is the set of algorithms or heuristics that provide the basis for parties to negotiate. Negotiation is a series of messaging rounds where the type and content of every message is determined by the history of the current negotiation session (*context*) and the immediate message received from the negotiator. Both negotiating parties work with partial information; they have complete knowledge of their local policy and resource requirements, but probably minimal knowledge of the opposite entity's. Therefore, after every round of messaging, both entities gain more knowledge about the other's policies and capabilities; this helps them to make better decisions. Therefore, any algorithm or heuristic that a negotiation entity employs must be used along with instantaneous (and incrementally gained) knowledge to engineer the protocol toward

the most successful result possible. The heuristics determine what *success* for a computing entity is. Such heuristics would likely be functions of the utility of the desired resources possessed by the other, and security/access control risks determined through the trust model. For either party, the gain of resource access must be balanced with the risks of local resource and policy exposure. Minimizing the number of negotiation steps, which might involve making bigger compromises, could be another heuristic. Decision making is the application of heuristics to current context and local policy.

The notion of a *strategy* combines the iterative application of heuristics into a coherent protocol. Such strategies could dictate both the pace and the result of an instance of negotiation. The notion of strategies has been investigated to some extent in *automated trust negotiation*, where a web client requires access to a resource controlled by a web server, and iterative release of access-control credentials results in a positive or a negative result. This work is more of a template-based negotiation, with the objective being to allow resource access or not, based on local policy. This technique cannot be used to negotiate with resources, reevaluate goals, or manage conflicts and context changes. Resource negotiation in my research is not limited to yes/no answers, but provides a range of compromise solutions. Still, the analysis of strategies, especially interoperable ones, is interesting and will be leveraged during the course of this research [Yu2001]. Such strategies could range from *eager* to *lazy* (as defined in trust negotiation [Winsborough2000]). The former would result in a negotiator completely exposing all its requirements, supporting credentials, resources possessed and policies governing how (and to what extent) access to those resources can be granted right at the beginning so that the protocol can terminate as quickly as possible. The latter would take the opposite course, being paranoid about any exposure or access. Interesting research questions could arise from this. For example, informing the opposite party about a local access control policy could expedite the protocol but might result in an inadvertent leak of information that could be abused by the other.

Strategies have been studied in subjects like game theory and utility theory from the fields of mathematics and economics [Owen1995][Fishburn1988], though in a static context. Adversarial game theory deals with situations where one entity tries to outwit the other within a set rule framework and with partial information. Utility theory deals with agents that have ideas about relative *utilities* of objects and situations, and engineers its strategy toward achievement of maximal utility. As we can see, policy-based negotiation could potentially be modeled along these lines. From a practical systems perspective, probabilistic or Bayesian reasoning could also be applied, though that is not a priority for this research.

In summary, I intend to apply trust and utility heuristics to the development of negotiation strategies and study the relative benefits of different types of strategies in identical scenarios.

4.6. Theoretical Issues

The negotiation model articulated in Section 3.3.1 provides a basis from which we can consider the problem in purely theoretical terms, though in addition we would need some stronger assumptions about the characteristics of resources, contextual parameters and policy rules. Though the success of a system research project is often measured by simulation and experimentation with practical situations, formal guarantees of results and proofs are extremely valuable and serve as guidelines for further system development. Two such research issues are correctness and completeness.

Correctness: When is the result of a negotiation *correct*, and how do we provide formal guarantees that a negotiation protocol yields a correct result (if more than one exists)? The notion of correctness is fairly simple to state; i.e., any modification of system state (granting of resource

access permissions, disclosure of data and adjustment of system behavior or security settings) must not come into conflict with collective local policy and heuristics. If policies are written in a formal language backed by a reasoning framework, such as first-order logic, it would be possible to give formal guarantees about negotiation results. For example, BAN logic [Burrows1990] is used to formally verify safety properties of security protocols. Similarly, the negotiation strategy, which includes tradeoff decisions and relative importance of policies, should be backed by a mathematical model, though this might be harder to conceive and analyze. It would be interesting to investigate whether negotiation heuristics can be viewed in purely logical terms, since our scenarios allow potential for policy conflicts. As a side issue, if efficiency considerations necessitate probabilistic reasoning, we can obtain a correct result only with a certain probability.

Completeness: Given an oracle or a centralized authority, an open system promising ubiquitous services would have knowledge about the resources, policies and requirements of both negotiating parties. Then, it could potentially run an algorithm that reasons with the collective policy and comes up with an agreement that would yield the best result for both entities. Such a best solution does exist; if equally "good" alternate solutions exist, a non-deterministic selection could be made, probably through human intervention. A negotiation protocol can be defined to be complete if it is guaranteed to find the best solution in a scenario with decentralized authority and private policy. Therefore, it is worth investigating whether or not such completeness can be guaranteed by a negotiation protocol and associated algorithms. It would hopefully lead to efficient schemes for reasoning and decision making with incomplete knowledge. If completeness could be proved, associated questions may also yield answers, one example being: "*what is the least number of negotiation steps that would lead to a satisfactory solution?*" Prior work in distributed algorithms and game-theory will certainly be leveraged, but realizing a truly complete negotiation protocol will probably be computationally infeasible, maybe even more so than AI search problems. Therefore, we may have to settle for a best-effort, heuristic solution.

In summary, I intend to analyze and give (maybe informal) correctness proofs for all negotiation algorithms and policy reasoning mechanisms used, since the lack of these could have adverse consequences in a real-world system. Completeness could be proved by an algorithm that takes a strategy and an ideal result as input, but the generation of an ideal strategy will be investigated only if time permits, since it will require non-trivial effort.

4.7. Systems Issues

This subsection deals with the core issues involved in the design of the policy management architecture and the basic mechanisms that enable interactions.

4.7.1. Negotiation Protocol and Message Types

The negotiation protocol must allow the realization of the model that was proposed earlier in Section 3. Continuing the flexibility argument from Section 4.2, ***we need to discover a small group of message types and a state machine that can be used in a domain-independent negotiation protocol***. The claim is that a small core group of messages, including requests, offers and policies, is sufficient for any application scenario. Interpretation of the message content is domain-dependent, because policy is local. Whether or not this negotiation protocol is too generic or too restrictive (which doesn't seem likely) is a question that will be answered when we start experimenting with real-world applications.

4.7.2. Resource Management

Solutions for embedding resources in physical spaces (or domains) and providing interfaces to discover and access them via a network link currently exist [Román2002][Waldo1999]. Management of such resources within a domain can be done in a centralized manner using any of the available solutions, and is not a concern in my research. The functions provided by the resource manager can be called by the policy engine, and the resource descriptions and allocation mechanisms encoded as policies within our database. Still, assignment of permissions for actual resource access ties the negotiation and the resource management schemes somewhat tightly. The relevant questions are: i) what forms can such access permissions take, and ii) how does the policy manager arbitrate resource accesses when they occur? ***Policy enforcement is probably out of the scope of this research, but we still need to ensure that access permissions resulting from negotiation are suitable to such enforcement.*** Between the tried and tested access control mechanisms like ACLs and capabilities, the latter seem to offer a more scalable solution in a dynamic environment, where the list of potential negotiators is unbounded, though revocation is difficult. Experiments will tell whether capabilities are suitable and adequate. Whatever kinds of access permissions we use, we need to embed cryptographic safeguards in them. It would also be an interesting exercise to investigate how policy and context information could be embedded in a tamper-proof manner into a capability (or equivalent token), which could be verified by the policy manager whenever an actual access is requested.

4.7.3. Performance

The overhead of performing negotiation must be hidden as far as possible from the users. Applications on a mobile device that must run seamlessly across domain boundaries also depend on the negotiation protocol performing within soft real-time constraints. The first step for a device is, of course, to discover and connect to a network, following which negotiation takes place.

There are two potential performance bottlenecks:

i) The size of the policy database, and the resultant slowing down (potentially exponential) of policy inference and resolution. ***The inference procedure must scale with the size of the policy set.***

ii) The negotiation strategy, which, without real-time heuristics, could take a number of messaging rounds to terminate.

In the first case, current Prolog compilers do use efficient indexing, pruning and retrieval techniques that make such systems fast in practice, but the search space that the policy engine reasons over still needs to be reduced. Pruning the search space in order to obtain *relevant* policies and state information during a decision-making process would be valuable. Any logical mechanisms that we use, such as forward chaining, backward chaining, conflict checking and truth maintenance, may also require some re-engineering to make them scale. One other optimization could be a policy *fast path*. This would be a front-end module for the policy manager or the system component that normally makes negotiation decisions and would be populated and cleared in the manner of a cache. Lookup in such a table would take constant (or *O(1)*) time, and it could answer any questions or return negotiation decisions that a full policy manager would. Such modules are frequently used in security systems like firewalls and intrusion detection systems. Context could determine what information would be stored in the fast path. Interesting questions that will be investigated are the following: what is the structure of such a table, can it handle all policy-related decisions, and if not, what is the class of decisions it can handle. The result of this research will be a fast path that can provide quick answers to queries and is also guaranteed to be correct. This implies that an answer received from the fast path

would exactly match the answer to a similar question posed to the full policy manager. On the other hand, incomplete knowledge may prevent the fast path from answering many of the questions.

In the second case, constraints (both hard and soft) on time or number of messages would be added to the heuristic that guides the negotiation strategy. In the absence of such constraints, negotiations could terminate quickly if *eager* strategies are adopted. These approaches will be experimented with in the course of this research.

4.7.4. Fault Tolerance and Reliability

The negotiation framework must maintain the integrity of the local policy engine, the consistency of the policy database, and must ensure the robustness of the negotiation protocol. ***The protocol itself must be tolerant to communication failures (like loss of network connectivity) and to the failure of the policy manager during a negotiation.*** Distributed systems normally use two ways to deal with failures: one is redundancy and the other is detection and recovery (or rollback) from failure. Making redundant copies of a policy manager could help, but would pose an additional burden on low-capability devices. Network failure is always a possibility with a wireless connection, and must be dealt with. The policy database could change dynamically with policy, and because of parallel negotiation sessions accessing it simultaneously. Such changes need to be as secure as database transactions. Therefore, the negotiation protocol must be robust in the same way as database commit protocols, so that both entities commit to any agreements, and possible mechanisms allowing rollback upon failure. Other problems that traditionally occur in distributed system, like clock synchronization, may not be relevant in negotiation, since both negotiators make independent decisions; however, some secure communication protocols may require timing guarantees. The initial approach for a protocol is outlined in Section 5; different failure scenarios will be looked into during the course of the research and appropriate solutions investigated.

4.7.5. Operation in Resource-Poor Conditions

Certain characteristics will be typical of pervasive computing environments and mobile devices. Most devices, especially mobile ones, are not really resource-rich. For example, it is obvious that a framework that is meant to run on a heavy-duty web server cannot run on an E-Watch. Small devices that need to run negotiation middleware, maintain policy databases and use security mechanisms locally and for connections are typically limited in CPU power (which may prevent devices from handling heavy cryptography), memory, interfaces and energy (batteries typically don't last long with wireless connections). Most networking in the environments of the future is likely going to be wireless, which is a shared, lossy medium. Therefore, the negotiation framework must be robust enough and adaptable enough to be usable by devices with low capability and inadequate interfaces, and work in conditions of intermittent connectivity. Fortunately there is a large body of research that has been done or is being done in such areas. Research in transcoding deals with adaptation of applications and content across platforms, taking into account both the rendering of content on local interfaces, and the system and network load such devices can practically handle. ***My research will address how system capabilities and constraints are specified in policy and incorporated in negotiation heuristics, and how the above-mentioned mechanisms can be used for dynamic adaptation.*** Intermittent connectivity is, of course, a fault tolerance issue, dealt with in Section 4.7.4. The Panoply framework allows us to use various optimizations such as maintenance of long-term agreements with known domains, causing negotiation to be relatively lightweight.

4.7.6. Integration with Panoply and Legacy Systems

A framework for managing system policy and performing negotiation will be designed and built in the context of the Panoply framework, described in more detail in Section 5. Research will show how a policy-controlled sphere can provide a scalable infrastructure with a rich set of functions that are necessary for ubicomp.

Since the policy manager is implemented as a Panoply sphere component, any device or domain that does not run the sphere software could be considered a legacy system. Still, policy managers provide useful functions that are independent of the spheres concept. Any two devices that have policy managers deployed can negotiate with each other. Making legacy applications and other device software work with a policy manager is probably an engineering question. A possible answer could be a module that intercepts network connections and system calls, and redirects them to the policy manager; this approach is used by the Conductor [Yarvis1999] and Panda [Ferreria2002] middlewares. ***The primary goal is to build a policy manager as part of the Panoply middleware; legacy issues will be examined and handled as required in the context of any application we choose for demonstration purposes.***

4.7.7. Context Awareness

The policy manager uses context information in conjunction with local policy to make decisions. It should, therefore, possess a means for tracking context information, which changes especially quickly in mobile and ubiquitous computing scenarios. Principal contextual parameters are location, time, and the state of the local environment (entities within, available resources). *Context sensors* keep track of each individual piece of context information and can provide information to, or be queried by, the policy manager. Much work has been done in the past few years in location tracking – GPS being one example. The local environment should keep track of state, which in our case, the Panoply infrastructure for each sphere does. *Sensor networks* could be used to sense any kind of contextual parameter change and communicate it to subscribers. Therefore, obtaining and tracking context information can be done by using techniques already developed or in the process of development, and is not a research issue per se. Use of context information for decision making is part of the role of the policy language and reasoning engine.

4.8. Beyond Two-Party Negotiation

The negotiation model described in Section 3.1.1 extends by induction to the *n*-participant case. Two different scenarios in multiple entity negotiation are 1-*n* (multi-session) and *n*-*n* (multi-party) cases. A good two-party negotiation solution could be extended to solve the more general problem, though the latter raises additional issues.

Multi-session negotiations are required in scenarios identical to client-server systems where a server handles parallel sessions with multiple clients. The server usually handles multiple clients by creating a separate instance of the protocol state machine for every connection, and in most cases, communication threads are independent of one another. Multi-session negotiation in ubicomp will involve a network policy manager dealing simultaneously with multiple prospective client devices owned by different mobile users. Negotiation with every client is based on the same shared knowledge base and policy. Therefore, all negotiations are interdependent. Any change in system state, goal reevaluation, or a different constraint arising as the result of one session, affects other sessions. Scalability is affected by such interdependence of sessions as well as load limitations on the device handling multiple negotiations. The goal of maintaining system integrity under dynamic conditions is identical to the two-party case; indeed, we could leverage indexing and search space pruning techniques used here to handle session interdependence.

Multi-party negotiation is the most general negotiation scenario in which a set of entities are negotiating among themselves, each entity aware of one or more of the other *n*-1 participants. Apart from the issues raised above, this kind of negotiation will have to deal with additional distributed system problems like synchronization and Byzantine failure.

In summary, the primary research problem is to ensure scalability by identifying the common elements in multiple negotiation sessions and considering load as an additional heuristic. Implementation of multi-session negotiation is one of my goals. Multi-party negotiation, on the other hand, opens up a whole spectrum of research issues, which will, in all likelihood, not be dealt with in my implementation, though I hope to give some insight to possible solutions in my dissertation.

4.9. Usability

Designing good user interfaces is not a goal of this research. On the other hand, ensuring the usability of a ubiquitous computing system and devices that use negotiation middleware certainly is, though it may be orthogonal to the problem of designing a negotiation protocol, and may involve AI. Users will be involved in setting policy, making their preferences known to their computing devices, changing such preferences when they wish and getting suitable feedback, or choices, from their devices when policy issues cannot be resolved automatically. Such feedback should be provided by a policy manager to the system user in the case of an irresolvable conflict or if it is faced with choices that cannot be compared without additional input. The extent to which user intervention is needed in negotiation depends upon policy settings, and is also a matter that requires investigation, since we don't have good answers.

A system will be usable if the middleware ensures that the users can do such tasks easily, or are given suitable hints by their devices in an *understandable* manner. For example, a user will not be concerned or even be aware of what it means for a service running on port XX to be vulnerable to a buffer overflow attack. He will understand warnings if they are given in more user-friendly terms and at a higher level, such as a chance of his files being deleted, or the fact that handing out a piece of private information could potentially allow anyone to access his email.

Identifying the objects and mechanisms that a user would be familiar with, the level of generality (or lack of specificity) that a typical user would be comfortable with, is an interesting problem in the context of policy writing and high-level language design. So is the problem of adapting feedback to the current context rather than displaying a vague general statement that confuses the user. How such information is rendered, either as text, audio or video, and the actuators that generate interfaces, is outside the scope of my research.

To summarize, the research issues I intend to work on and make contributions to are as follows:

- ✓ Design a negotiation protocol, concentrating on the following aspects:
  - Flexibility and extensibility
  - Robustness and fault tolerance
  - Performance and scalability issues (both two-party and multi-session) associated with policy management during negotiation
- ✓ Develop models for making risk-benefit decisions on the basis of trust and utility
- ✓ Develop strategies for negotiation that can be implemented using the protocol
  - Make decisions using the trust and utility models applied to local policy and current context
  - Analyze correctness and completeness of strategies associated with those models
- ✓ Design a policy language and ubicomp ontology that will be based on logical reasoning mechanisms and which will support:

- Specification of resource, security and environment constraints
- Users' ability to write and manipulate high-level policies

Other issues which I have mentioned but which are outside the scope of my research are as follows (these are orthogonal issues being worked on by other researchers):

- Building of discovery of device communities (Kevin Eustice's research)
- Resource management frameworks, interfaces and access mechanisms (for which various mechanisms exist), gathering of context information (again, mechanisms exist)
- Design of user interfaces with information rendered in a suitable manner (my research will be limited to identifying the suitable information)

5. System Design and Implementation Approach

In this section I outline the way I propose to design and implement a prototype negotiation framework. The negotiation framework can be logically broken into three parts:

i) negotiation protocol architecture
ii) policy language and reasoning framework
iii) negotiation algorithms and heuristics

These three parts are functionally independent of each other, proving the necessary flexibility; for example, one could replace the negotiation heuristics and algorithms, keeping the protocol and the policy language intact. These functions are performed by what we have already referred to as a *policy manager*, which is structurally a middleware, residing above the operating system layer and below the application layer. It provides an interface for interaction with local applications, and for direct communication with the middleware on an external device. In a Panoply sphere implementation, the policy manager is a module within a sphere manager that controls all the activities (internal and external) of the sphere that hosts it (see Figure 1).

The core functions and mechanisms of each identifiable module are part of the initial prototype, which I describe in this section, so that the system can be evaluated in its entirety. Appropriate hooks to allow debugging, adding plug-ins and replacing modules in the future will be put in place. With further progress in research, thought and intermediate performance results, the framework will be augmented with more powerful mechanisms, more "intelligence," and support for a wider variety of applications. Some of the implementation targets have already been met and the current status is outlined in Section 6.1.

First, I describe in brief the design of a Panoply sphere infrastructure, which will serve to provide perspective to the design choices I make for a policy manager that facilitates negotiation.

5.1. Panoply and Spheres of Influence

The Panoply architecture facilitates the building of device communities or groups (*spheres* in our parlance) which can act as one when interfacing with the outside world; a single device is a unit sphere. The *sphere manager* manages all activity and relationships within a sphere, and as such is the equivalent of an operating system for a sphere. Architecturally, within every device, the sphere manager lies above the operating system and below the application layer (see Figure 1). All inter-sphere and intra-sphere communications follow an event model. An external event interface serves as the security boundary for a sphere and for interactions with external spheres, while an internal event interface is used to communicate with child spheres and to interact with applications. The sphere manager contains both a relationship manager and a policy manager. The latter manages local policy, answers queries and negotiates with external spheres; it interfaces with other spheres and applications through special *policy events*. The functions of the sphere manager, apart from those handled by the policy manager, is research that

is being carried out by Kevin Eustice of the LASR laboratory. Figure 1 describes the overall Panoply architecture as represented in Kevin's Ph.D. prospectus. Kevin's research and implementation is currently at a stage where it can support the kind of policy management that I am building; details of the status are given in Section 6.1.

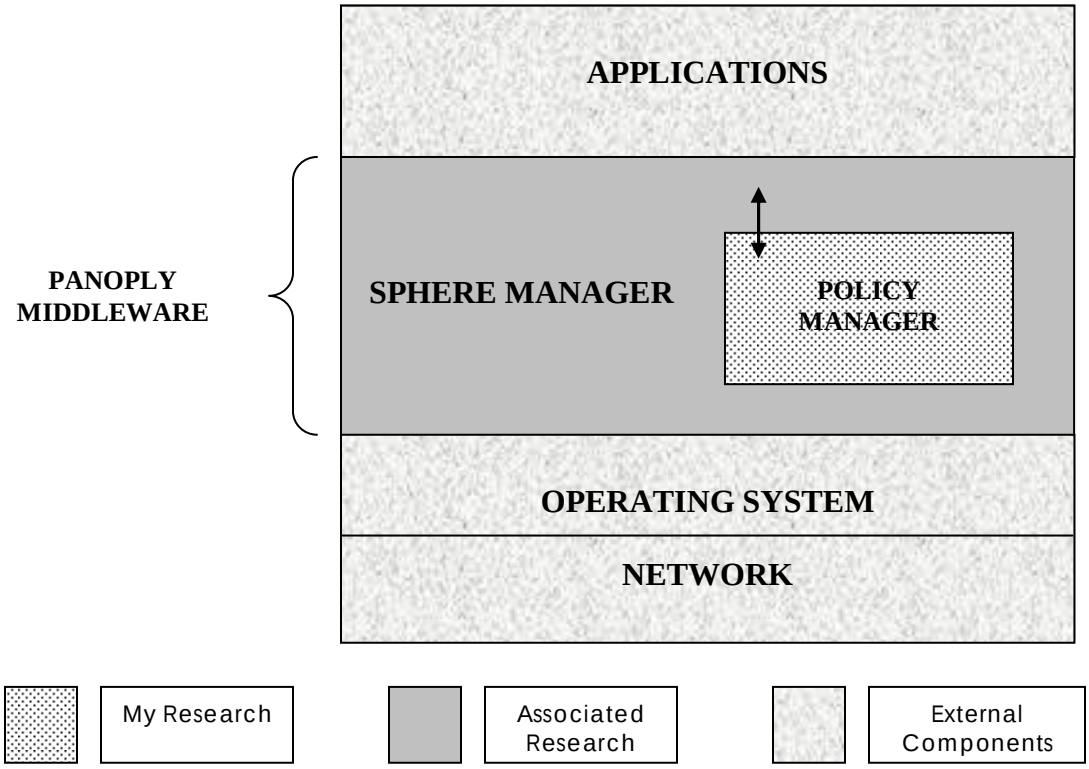

Figure 1: **Overall System Architecture**

5.2. Negotiation Framework Architecture

The policy manager architecture (Figure 2) consists of three layers (from bottom to top):

i) *policy engine*, or *back end*, that maintains the policy database and provides mechanisms and functions to manipulate and query the database
ii) *controller* (middle layer) that controls the way the negotiation protocol proceeds by utilizing the mechanisms provided by the policy engine
iii) *front-end* responsible for interfacing with applications, operating system libraries and external devices

The policy language and reasoning framework are almost completely part of the policy engine, whereas the algorithms and heuristics, as well as the security model are mostly part of the control layer. There could be some overlap of these logical functions across layers, and the extent of this will be determined as the research progresses.

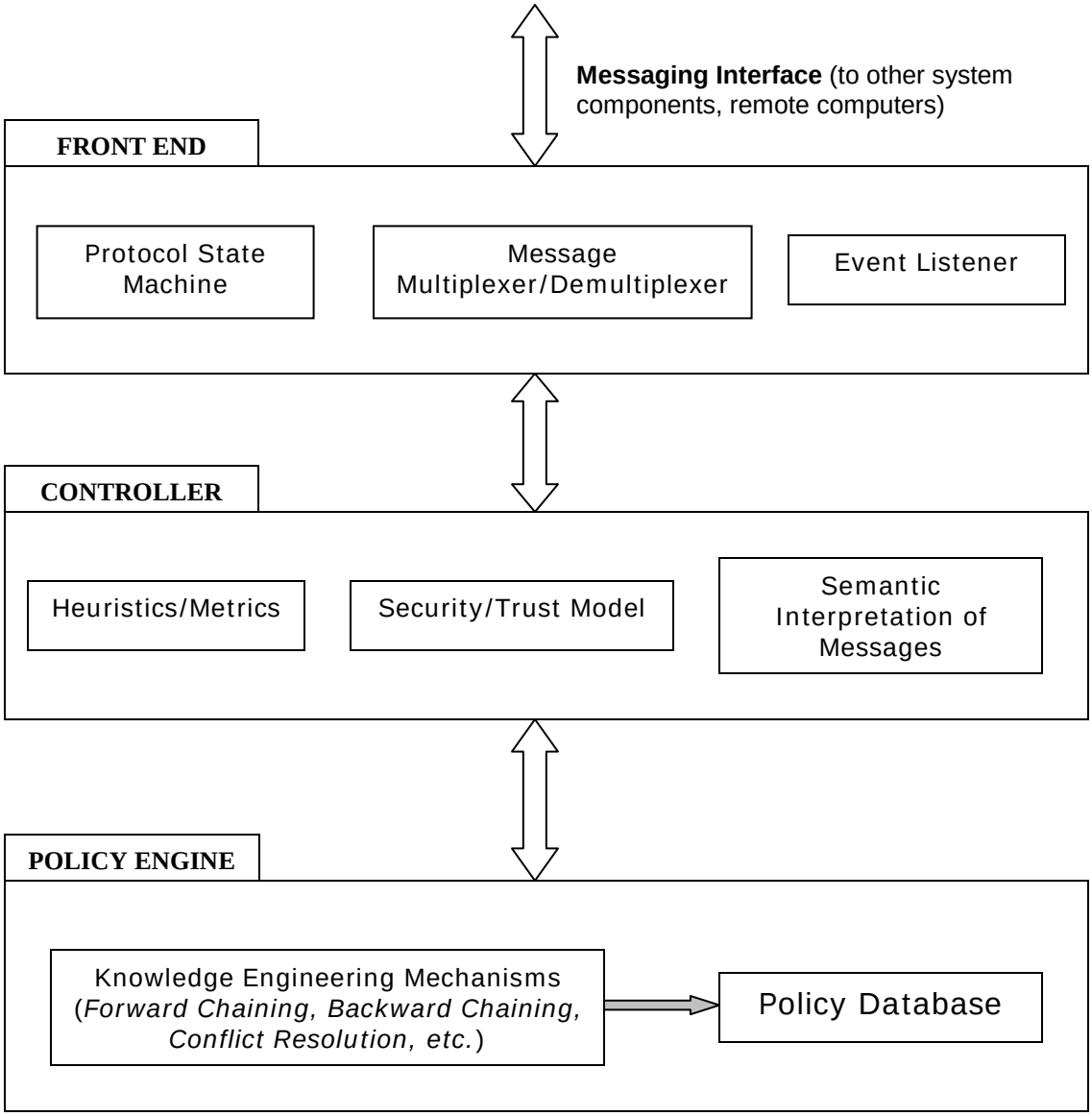

Figure 2: Functional View of the Policy Manager

5.2.1. Negotiation Interface/Front-end

This layer is the outer shell of the policy manager that acts as its interface to other modules. If the local environment, or any application running on it, wishes to query the policy manager regarding a system state or policy issue, it interacts with this component. Negotiation messages are sent to and received from this component, which functions as the interface and message multiplexer/demultiplexer. It can subscribe to and listen for events, of which context change events are particularly important.

A negotiation protocol must maintain state information for every negotiation session that is currently under way. The current design supports every entity running a state machine to negotiate with exactly one remote entity. The state machine that processes the messages at a high level runs on this topmost layer. The semantics of handling message content is left to the middle,

or control layer. Multiple entities can be negotiated with simultaneously using a multi-threaded approach; any conflicts or side effects are taken care of in the lower layers, since we have a shared database. The need for an integrated state machine that can process multiple clients is not obvious at this stage. If tasks such as maintaining the knowledge base free of policy conflicts and performance optimizations require an integrated approach, we will investigate it at a later stage.

The negotiation protocol has a few high-level message types:

- *requests*: contain desire for access rights, permission to do some tasks that involve using the other entity's resources, or information owned by the other entity.
- *offers*: contain replies with reference to any request(s) made by the opposite entity; may contain prior requested access permissions or data items; our current negotiation strategy limits offers to things asked for earlier, but the enhanced model will add more flexibility.
- *policy*: contains policy constraints that prevent opposite entity from getting its request granted, or must be satisfied by the opposite entity; also includes any obligations that are part of any request granting policy.
- *termination*: signals that the entity is going to terminate negotiation.

These messages could contain other supporting information apart from what is indicated by the type; supporting data, certificates, or any information for interpretation of a request, offer or policy could be appended. A protocol state machine runs at both ends of the negotiation, with state transitions being triggered by the arrival of a message and dependent on its type. Communication is peer-to-peer, and either party can initiate negotiation. The state machine is blind to the source of the messages coming from a remote entity, and can implicitly handle multiple clients (though the current implementation uses a multithreaded approach). It is expected that the nature of this machine will change with further progress in my research. For this reason and for the sake of brevity, I omit the details of the protocol in this prospectus (the full specification will be present in my final dissertation).

5.2.2. Policy Engine/Back-End

The bottom layer of the policy manager provides knowledge engineering mechanisms. Given a database containing facts and rules, the policy engine can provide views and answers in response to queries issues by a higher layer. The reasoning framework is built into this engine, which runs algorithms that obey the semantics of the policy language and the underlying logic. It runs mechanisms that support adding and removing policy rules, or modifying them. It provides ways of maintaining the consistency of the database, and to detect potential conflicts. Some examples of such mechanisms are forward and backward chaining. There are other standard AI techniques that would likely need to be employed, like truth maintenance and reasoning with default logic.

In order to process queries efficiently, the policy engine must also run efficient indexing and retrieval algorithms. The infrastructure I am currently using to develop the policy engine and language is SWI-Prolog, which does support some mechanisms for these, and these will be used at the beginning of the implementation. Later on, indexing and retrieval techniques may have to be augmented based on how negotiation heuristics and algorithms need to control them.

5.2.3. Controller/Decision Manager

The middle layer is the bridge between the outer shell and the policy engine. The main function of this layer is to control the negotiation protocol. Messages sent below to this layer are interpreted and processed using the mechanisms provided by the policy engine. The algorithms and heuristics that are necessary to interpret the semantics of the messages and to choose appropriate knowledge engineering mechanisms are employed at this layer. It will usually be the

case that various disparate knowledge bases will have to be combined and a relevant portion selected which is then processed by the policy engine. For example, current context information and relationships with neighboring entities in the local environment must be captured through other means (in Panoply, this will be done through sphere events). This could then be resolved with the policy of the local device to get the state information and the behavior rules that are relevant to the current query being processed.

Examples of ways the controller could invoke policy engine mechanisms are: when a request is made by a remote negotiating entity, it would require backward chaining through the policy database to generate an answer or detect a policy conflict; when some information must be added to the policy database, or something needs to be modified, forward chaining mechanisms could be employed.

5.3. Policy Language Design and Implementation

A good policy language framework must allow easy addition, removal and modification of policy statements. Therefore, anyone who writes policy must be able to specify what the desired behavior and system facts should be and not how the system should interpret the facts or exhibit such behavior. Declarative programming languages provide exactly this paradigm. Added to that is the requirement that the policy language semantics must be based on a sound logical framework. In addition to this, theoretical functionality must be balanced with performance constraints in a practical system.

First-order logic seems to fit the requirements for expression of policy in ubicomp. Full first-order logic support requires the use of theorem provers, which are not very efficient. A balance can be struck by using logic programming languages. Such languages impose certain restrictions; they handle only horn clauses, the only means of inference is backward chaining, negation is proved by failure, (i.e., if a query fails, it is considered unsatisfiable or false), and cyclic policies could generate infinite loops. On the other hand, logic programming languages offer much better performance than more comprehensive logic engineering tools. They have standard built-in compilation and indexing techniques that make them fast, with modern languages processing up to several million LIPS (logical inferences per second) [Roy1990]. And compared to other programming paradigms like imperative and functional languages, the logical basis and the declarative style of programming are distinct advantages.

Prolog is the oldest of the logic programming languages, and the one to have undergone the most serious transformations and improvements to the point where it is practical enough to use it in current large-scale expert systems. Its use has been demonstrated in many practical scenarios, including the detection of conflicts in a role-based access control framework [Schaad01]. SWI-Prolog [SWIProlog] is a comprehensive open-source Prolog framework that not only allows direct programming in Prolog, but also provides various APIs with other languages, like C and Java and RDF. There are a number of meta-predicates and constructs that it provides that enable higher level manipulation of the predicates defined in a Prolog program; incidentally, these can be used to reason with the knowledge base using techniques other than just backward chaining. Of course, these won't be the most efficient ways of doing those, but are satisfactory for a prototype. A big advantage of using SWI-Prolog is that the code is open source. Therefore, we can modify the language if further research indicates that Prolog needs to be modified to fit certain classes of applications or policies, or if we need to make further ontological commitments for location, time, etc. In the current implementation, the Prolog engine is accessible through a Java API.

Prolog by itself provides a syntax that is intuitive for writing policies, though it is flexible enough that one can use it for almost any purpose (we get policy rule parsing and semantics for free). The ideal realization of a policy language will be a set of high-level predicates and

constants (that are human-understandable), which policy-writers can use. But as long as we can describe everything of interest in our policy language, presenting a higher level interface may just require a straightforward syntactic transformation. Ubicomp ontologies like SOUPA [Chen2004] provide a good ideal launching pad.

5.4. Description of Resources and Properties

The language used to describe resources, properties and contextual information must be understandable in a wide variety of computing environments and applications. The *semantic web* leads the way toward a standard in this respect, being promoted by the W3C and backed by some of the top researchers in the field. Though a standard is still some distance away, some of the contributions and recommendations are already followed by a large number of administrative domains and application designers, such as the RDF model for naming and property structures, as well as a commonly understood XML (E**x**tensible **M**arkup **L**anguage) syntax. In the long term, this research project will deal with resource description using the RDF and XML frameworks, and policy descriptions will incorporate RDF standards as far as possible. Negotiation messages will also, in the long run, be encoded in XML. Therefore, even a proof-of-concept research project can be used and enhanced directly in other environments instead of requiring heavy redesign.

5.5. Security Model

The policy manager will be able to support various security, access control and proof of identity mechanisms that are commonly used and which have proved most effective, such as virus scanners, certificates and so on. The policy language should be able to support the description of the constraints associated with these mechanisms, verification of the security properties and assessment of the risks associated with non-possession of credentials or possible presence of viruses. These functions will be used as primitives for the policy engine to make runtime decisions. The mechanisms used in the project will be selected on the basis of the different applications that are used for demonstration; these will be decided at a later date.

As described earlier, security is the primary constraint on the basis of which devices negotiate. The security mechanisms help to assess the level of security present and the risks associated with any action. This must be tied closely to a trust model that allows assignment of trust levels to entities, which is a measure of risk associated with any permissions or access granted to it or to expose private information. Negotiation helps to build up such proof so as to constantly update this trust level. A good place to start with an access control model seems to be the GRBAC [Covington2000], to specify roles for entities, objects (resources) and environments (context). Delegation will be supported, as in the DRBAC model [Freudenthal2002] as it assists in building proofs and assessing risks while building a web of trust. The exact logic for the trust model, and the rules for trust inference are not clear right now, and this will be a key research focus in the near future.

5.6. Initial Negotiation Approach

The policy manager is being designed as an independent component of the Panoply framework. Initially, therefore, the negotiation protocol serves some of the fundamental sphere functions, such as one sphere *joining* another, or adding a member relation. This basic version of negotiation requires the specification of negotiation targets to be decided beforehand; these could include obtaining permission to access resources, establishment of relationships, queries, and subscription for events. The policy rules are designated public and private in order to let the

manager make suitable decisions. The messaging consists of requests, counter requests (for resources in exchange, or proofs of identity, etc.), offers and policy disclosures. Since the basic version of Panoply does not have resources, policies generally consist of system state descriptions and access control rules. Initially, the policies are written in Prolog, and many could describe low-level mechanisms. In the long run, the aim is to create a higher-level view that lets ordinary users write policy.

The main tasks are currently done in the front- and back-ends. Research in the near future will concentrate on heuristics that can be built into the control (middle) layer. These include trust models that can figure out (through some evaluation function) risks of interaction using certain mechanisms and allowing access to certain entities. I will also look into a resource utility model, which, combined with the trust model, enables the system to make risk/benefit analyses. This will allow compromises in terms of offering alternative resources, or different levels of quantitatively measurable resources. Experimenting with these in different application scenarios will be enlightening.

5.7. Interfaces

The policy manager will interface with other components of a sphere through *event interfaces* that allow clients to register for events, get and post events. Negotiation messages are encapsulated as events that are currently implemented as Java *serializable* objects and will, in the long run, be encoded in XML, which is portable and enjoys wide acceptance. Queries about sphere state or policy are also made through the event interface.

User interfaces will provide ways for system users and administrators to make changes in system policy, and to make high-level decisions in situations where it is impossible for the policy manager to choose from a list of equivalent options.

5.8. System Specifications

The Panoply infrastructure is currently being implemented under Linux, with the middleware comprised of the sphere and policy managers written mostly in Java (version 1.4), which is easy to use, portable and provides a good networking API with support for secure communication protocols like TLS. The policy engine is based on the Prolog logic programming language, for which we are using SWI-Prolog version 5.4.7, which also provides the Java/Prolog functional API.

6. Dissertation Plan and Schedule

The design, implementation and evaluation (both for testing functionality through applications, and performance) of a policy manager will proceed in iterative cycles. At the end of each cycle, we will obtain clues from the results of the previous cycle and from additional, complementary research. These clues will indicate the direction that further research should take and the features that should be added when designing the next version of the policy manager, which would make its behavior more intelligent and more widely usable. The integration with Panoply will proceed in parallel, and evaluation will be done in that context. The ultimate target is a negotiation framework that can be applied to scenarios described in Section 3, especially the Starbucks example. The result will be two entities that can negotiate on the basis of their policies, trust and utility models, and can demonstrably reach a working agreement that minimizes risk and

maximizes resource requirements. Such agreements should also demonstrably change with context; these context changes can be arranged through Panoply applications.

6.1. Implementation of a Basic Infrastructure

A prototype policy manager that supports negotiation based on the architecture described in Section 5 has already been implemented and integrated with the Panoply infrastructure. It has a handle to the local sphere manager (see Figure 1) and communicates with applications and external policy managers (located in other spheres) through *events*. Currently a number of desired features are in place in the front-end and the policy engine. The former runs the protocol state machine, handles multiple negotiations in a multi-threaded manner, and multiplexes and demultiplexes messages. Various policy engineering mechanisms for inference and resolution are built into the policy engine, which manages sphere facts and rules in a Prolog database using predicates and clauses. Prolog queries are made through a Java API provided by the JPL package, which is part of the SWI-Prolog project. Through this API, the local policy database can be queried, and various meta-predicates provided by SWI-Prolog enable the control layer to select policy rules and determine the choices to be made. The control layer does not have a trust model or a utility model built in, and as such employs a simple, yet cautious, negotiation strategy, using only the stated policies in the database, through backward chaining mechanisms provided by Prolog. In response to received requests, it makes decisions about sending offers, or computing counter requests in case current policy cannot allow an offer. Alternatives are computed and proposed if certain requests are denied. Requests, offers and policies are formulated as Prolog queries, though this may change later to an XML/RDF representation. Designated predicates and clauses are used for standard interpretation both at a higher level (by the controller) and across domains. Policies can be added, modified and deleted during runtime by the applications. Sphere state changes can be discerned through sphere events, though many more useful context hooks need to be added. Remote spheres can also query one another using the negotiation mechanism, and answers are provided based on access control policies specified at either end.

The main task that the policy manager currently performs is to negotiate with a prospective member of a sphere and decide whether or not a membership request may be granted. The policy database contains information about the sphere state and its access control policies.

Let us revisit the Starbucks example. The infrastructure implemented so far can handle the basic messaging that goes on between Starbucks and Bob; requests for bandwidth, email addresses and anti-sign credentials can be framed, as well as offers of credentials signed by Verisign. The current negotiation strategy only supports alternative requests and not alternative offers, which would limit the Starbucks network to approving or disapproving the original bandwidth request. Also, though policy statements and obligations can be communicated, they cannot currently be interpreted in the proper context and used for making decisions. The current design is also not backed by trust or utility models, and so individual access and release rules have to be specified explicitly for all the objects that are being negotiated by Bob's PDA and Starbucks network. To summarize, Bob's PDA will gain membership to the Starbucks sphere, with its original bandwidth request approved or rejected based on Starbucks' access control rules.

6.2. Evaluation of System Within Panoply

The basic policy manager described above will be evaluated as a part of Panoply. System behavior will be observed in the context of various applications: a simple scenario when a laptop enters Boelter 3564 and wishes to become a member of the LASR sphere (wireless network); an interactive fiction gaming application; a LACMA deployment spheres representing galleries and individual paintings, where a user carrying a personal device gets customized experience based

on which sphere(s) it is connected to. Simple access control policies governing sphere (or network) membership based on credentials (pure identity, certificates, delegated credentials or vouchers) will be written. System performance (i.e., response time or overhead of going through negotiation before devices can freely interact) will be evaluated, and the results of negotiation compared against original requests. The above scenarios will show that restricted negotiation using a limited policy language works, though not in a very flexible way. We can also demonstrate how spheres can generate requests, counter-requests, offers and alternatives based purely on specified policies. Other goals that still need to be met are specified below.

6.3. Policy Language and Engine Enhancements

The evaluation in Section 6.2 will also give us an idea as to the adequacy of using basic Prolog as a policy language. A survey will be done to decide on an ontology and a list of high-level objects that the policy language needs to support, which can be understood across domains. Apart from adding language constructs, we will also examine what the currently used Prolog engine lacks as far as reasoning mechanisms are concerned. Features such as "negation by failure," asserting negative facts and rules and lack of a default policy or truth and consistency maintenance system will be addressed. The result of this phase will be a more complete and widely applicable policy language and reasoning module.

6.4. Development of Security/Trust Model

The security and trust models will be developed during this phase. This model will leverage existing research, both theoretical and practical, and be incorporated into the negotiation framework (specifically the control layer). Logic for dealing with trust will be developed. As outlined in earlier sections, this trust model will enable an entity to evaluate the trust level of the negotiator, most likely in relative terms, based on local policy, context and negotiation history. It will guide the policy manager in the selection of the specific policy to be applied, reevaluation of goals and strategy. A comprehensive survey of currently used security and access control mechanisms (like packet and software scanners, certificates, delegations) will be made and will be mapped to the trust model dynamics.

6.5. Development of a Utility Model

The negotiation framework will be augmented with a resource *utility* model that provides a basis, like the trust model, for negotiation of resources in a set of give-and-take operations. The policy language should be made to allow specification of resource quantities (e.g. *x* amount of network bandwidth). Decision making from utility and game theory will be leveraged to develop strategies for negotiation. Different actions performed by a system will be studied in order to model risk and benefits of performing such action, and these observations will be leveraged in the combined trust and utility model. System state, persistent knowledge and user preferences (part of policy) will be used in the model and employed in risk-benefit computation.

The augmented policy manager will be evaluated on the basis of the security and access control functions it can perform. We will experiment with scenarios that have critical security requirements (or spheres with a stringent security policy), and observe whether the security model protects the local systems from policy violations.

6.6. Generalized Policy Manager

The working environment (spheres) will be augmented with resources. Policies will deal with the presence and state of various resources owned by a system, or services exported by it. They will deal with the constraints that are imposed on usage of such resources. The policy language will be augmented to add support for expression of resource identities and properties in a universally understandable language –RDF/XML being the most likely. The controller layer of the policy manager will need to manage multiple separate policy databases describing local policies, policies for applications running on the system and the contextual information, and grab a view (or a subset) that can be used to make immediate policy decisions (because the complete database will be too large). Indexing and retrieval mechanisms used by the SWI-Prolog policy engine will be reviewed and augmented if necessary. The controller will be augmented with the trust and utility models so that it can do more flexible negotiation. Currently, spheres possess ways to sense their locations and map them to *regions* that have attached semantic meaning. For example, association with a wireless LAN whose SSID is "CSD" could indicate presence in Boelter Hall with a high probability. If other contextual parameters need to be discerned, based on whether or not they create richer application scenarios, suitable support will be added.

6.7. Evaluation of Policy Manager

The generalized policy manager will be analyzed both theoretically and performance-wise. The theoretical analysis will involve completeness and correctness studies as outlined in Section 4.6. Strategies will be experimented with and interoperability will be studied. For practical evaluation, applications will be developed that make use of resources and private pieces of data, and in which contextual changes are dynamic and prominent. Critical and semi-critical access control and security policies will be added. Performance measurements will be carried out in the same way as in Section 6.2, and negotiation results will be evaluated against original goals in order to measure both the success percentage of the participating entities and the undergone risk. The exact performance metrics are not clear right now, and will be developed along the way. Such metrics may also provide general guidelines to other researchers working in this field.

6.8. System Optimizations

Performance results will, of course, guide the way the negotiation framework is designed and implemented. Even from a theoretical point of view though, a comprehensive inference procedure is a heavy operation, both in terms of time taken to perform and also in terms of system load for devices that possess fewer resources. Therefore, this phase will involve engineering a *fast path* (based on a design outlined earlier in Section 4) that acts almost like a cache for storing relevant policies which can be looked up potentially in constant time. This component may be added even earlier in the schedule depending on intermediate performance results. Performance comparisons will be made between the middleware with and without this fast path.

6.9. Miscellaneous Issues and Dissertation Writing

There are other research issues that have been mentioned in this document, such as multi-party negotiation, which I will work on, time permitting. That, and other issues such as probabilistic reasoning with policies will certainly be a part of an extended work section in my dissertation that will outline the benefits provided by this research to the field specifically and to the world at large. The dissertation will also produce a holistic picture of the entire body of work

I have proposed in this document. The related work section will be suitably updated and will be differentiated from my work.

6.10. Timeline

Milestone	Completion Date
Basic Policy Manager	July 2005
Evaluation of Basic Policy Manager	November 2005
Policy Language Enhancements	December 2005
Security, Trust and Utility Models	March 2006
Generalized Policy Manager	May 2006
Evaluation of Generalized Policy Manager	August 2006
Optimizations	October 2006
Writing Dissertation	March 2007

7. Related and Complementary Research

There is a significant body of research in the design of negotiation protocols, policy languages and access control frameworks for open systems. Approaches toward providing solutions to the core interoperation problems of service discovery and dynamic resource management have also been proposed in various frameworks, especially in the ubicomp smart spaces projects. In this section, we take a look at the contributions of these projects, how we can leverage concepts, and how these research efforts fall short in our problem domain.

7.1. Negotiation Protocols

The work that bears the closest resemblance to my research is *automated trust negotiation* [Winslett2003], through which web entities (typically client-server, but the model applies to peer communication as well) can establish trust in an automated fashion, the objective being access to a guarded resource. The ultimate result is typically a yes/no answer from the resource owner to the requestor. The negotiation protocol involves request and exposure of sensitive credentials evaluated using per-credential access control policy rules. The TrustBuilder project [Winslett2002] implements trust negotiation as an extension of the TLS protocol. Trust negotiation is a more flexible way of doing access control, where entities can control what private information is released at fine granularities, though it is only a special case of general resource negotiation. The policy language is not very expressive, being limited to Boolean expressions of credentials, which are sufficient only for scenarios that involve access to a particular resource that a client can identify. Policy engine reasoning simply involves checks for presence of credentials or determining which credentials need to be requested in order to satisfy the relevant access control policy. The lack of a semantic and reasoning framework in the policy language, as well as support for a wider variety of description, prevents it from being used in the wider ubicomp context. ATN does not solve the more general problem dealing with simultaneous discovery and negotiation of multiple resources; the goal is always fixed here. Also, in ubicomp, trust is not restricted to the presence or absence of credentials but can have a much broader definition where

risks can be analyzed using any subjective measure. Overall, adherence to rigid policies and lack of context-awareness, and from a practical point of view, being closely tied to TLS (which is too restrictive for the application scenarios we target), are drawbacks that prevent ATN from being used as a basis for negotiation. On the other hand, TrustBuilder provides valuable clues that I can use in my research, such as a credential-based trust model, policy language requirements and policy and negotiation strategies based on policy and trust information.

PeerTrust [Gavriloaie2004][Nejdl2004] advances TrustBuilder concepts and applies a more powerful model for trust negotiation in the semantic web rather than in the existing Internet. A distributed logic programming language based on Prolog provides more expressiveness as a policy language. Rules can be signed (equivalent to certificates) and their evaluation delegated to another peer, thereby enabling distributed and cooperative security. The use of logic programs for policy description validates my approach outlined in this prospectus. Apart from this, PeerTrust suffers from the same drawbacks as TrustBuilder, namely a fixed goal, lack of context-awareness, rigid policies and trust building restricted to exposure of cryptographic credentials. Both these systems fail to consider inter-dependence of multiple resource and security constraints, because they target explicit and addressable connections on the Internet rather than ad hoc ubiquitous connections. PeerTrust is still the closest working system to our own that we can find, and the first prototype (iteration) of our policy manager is quite similar to it, though it needs to be augmented with trust and utility models and lower dependence on trusted authorities.

Negotiation protocols are also used on the grid [GRID] to make use of services hosted on a remote machine that is not part of the same administrative domain. Grid negotiation typically involves matching resource owner and consumer preferences. The negotiation steps consist of proposals and counter-proposals, which are evaluated against resource utility functions (specified in terms of maximum and minimum values) [Lawley2003]. Agreements are reached when no higher-utility counter proposal can be generated and basic constraints are met, otherwise the result is failure. Negotiation strategies could include heuristics such as the number of messaging rounds and could be guided by game-theoretic algorithms like Faratin's [Lawley2003], or by the use of genetic and evolutionary algorithms [Chao2002]. SNAP [Czajkowski2002] is a service-level agreement protocol for the grid that is similar to a resource allocation protocol, and is agnostic of the resource or application type. Policies are private, and clients make requests and counter-requests for *level* of a service till the server is able to satisfy or reject the request. Grid protocols offer valuable guides as far as negotiation heuristics and resource utilities are concerned, but they do not consider security constraints and service discovery. The grid also does not require expressive policy languages or reasoning mechanisms, since only a single resource is being transacted. Lastly, these protocols are employed in static situations, and cannot be used in scenarios that involve mobility and context changes. Modeling of utility and risk is also an important problem in its own right, and these systems do not provide any clues in this area.

Options to *negotiate* are incorporated in a large number of automated agent-based protocols, typically involving client-server transactions, a number of them having been proposed as RFCs and some standardized. DHCP, *block size option* negotiation in the TFTP protocol, and the RSVP protocol for QoS negotiation in IP networks are representative examples. In each of these, an ideal target is defined and known to both parties, with the final result typically being a yes/no from server to client. The key decisions are made by servers, and policies are known to both parties. These frameworks contain a common negotiation protocol shell but do not offer much in the way of the dynamic, flexible and widely applicable negotiation that we aim to do.

7.2. Policy Languages

The policy language that comes closest to the needs of this research is Rei [Kagal2003a][Kagal2003b], which is targeted specially toward pervasive computing. The

designers of Rei correctly make the claim that a policy language for ubicomp must have well-defined logical semantics because the scope of facts and constraints that it must support is huge and that it needs to be domain-independent. Older languages, unlike Rei, treat rules as independent policies rather than as part of a system Only with languages like Rei can we be confident of detecting and resolving system-wide conflicts. Rei is based on what we can discern as first-order semantics augmented by deontic concepts of obligations, permissions, prohibitions and dispensations. It supports specification of actions, action classes, speech acts like requests, offers, delegations and revocations, as well as meta-policies like modality (e.g., rule A overrules rule B) or priority for conflict resolution. In addition, we can describe common resources, entities and constraints, as can be done with most other languages. Some features that we would like are absent however. Inter-resource constraints, support for contextual description which would let us determine specific policy instances, and support for high- to low-level policy translation is absent. In its current state however, it is probably the best language for us on which to leverage our policy negotiation framework.

Ponder [Damianou2001], a declarative policy language for specification of a distributed system security policy, deserves mention as an early piece of research in ubicomp policy, though it falls short of Rei in some respects. One can specify delegation policies, entity roles, application groups, constraints and obligations. It supports conflict resolution using meta-policies and policy enforcement through the triggering of foreign functions. It suffers from the same drawbacks as Rei, and in addition is an object-oriented language rather than a semantic language. Datalog also deserves mention as a logic programming language that has been used to manage database constraints, though it is more restrictive than Prolog. Keynote [Blaze1999] was one of the earliest policy languages used for trust management and access control in an open system. The language allows specification of credential types and instances, associated actions and state constraints, which could be used by a compliance checker when validating an object access request. It does not support context-awareness, meta-policies or deontic concepts and actions. It is too closely tied to entity names and credentials, and has been superseded by languages that are more suitable for ubicomp, though the concepts proposed in Keynote still remain relevant.

Portfolio and Service Protection Language (PSPL) [Bonatti2000] is a language that was designed to support trust negotiation (see Section 7.1). *Services* offered by servers are distinguished from information possessed by clients, like cryptographic credentials and plain-text data declarations, collectively called a *portfolio*. The language offers support for description of service classes, subset and domination relationships (which can be used to specify high- and low-level rules and their relation), state information (both persistent and per-negotiation). PSPL also provides valuable clues for evaluating requests and release criteria for private resources as well as policy filtering. The syntax, and the evaluation of rules is Prolog-like, and so some constructs that aren't directly supported, such as more complex trust relationships, delegated credentials and trust chains, could be added without significant difficulty. Like other non-semantic languages, rules are associated with resources and so inter-resource relationships and complex security relationships cannot be handled. Specification of general policies that can be adjusted with context, deontic concepts and meta-policies are not supported. Languages for trust negotiation like PSPL and DTPL [Herzberg2000] are monotonic (do not handle negative rules), which works fine in that domain but are drawbacks when negotiation needs to consider alternatives and tradeoffs.

Other policy languages, based on XML, have been designed for access control on the web, such as IBM's Trust Policy language [Herzberg2000], X-Sec [Bertino2001] and XACML [Lorch2003]. The former two support description of credentials and credential types, role types (in TPL), entities and attributes. Policies are used by servers to infer trust information through a more complex procedure than Keynote, though support for deontic concepts and different types of actions are missing. XACML supports more expressiveness in terms of groups of resources, protocols, actions and authentication mechanisms, and allows specification of conflict resolution

rules, but is not suitable for our aims like the other two languages as it lacks a logical reasoning back-end.

Most of these languages, apart from Rei, are applicable only in a few select domains, where specification is closely tied to enforcement. They typically consider policy rules as isolated statements to be evaluated independently or in specified groups, rather than as part of a knowledge base. In my research, I take the approach of building bottom-up from a language with loose semantics and a reasoning mechanism (like Prolog) rather than making syntactic choices (like XML) and adding new constructs and logical reasoning mechanisms.

Web (or semantic web) ontologies will be leveraged in this research as a way of making the policy language and negotiation methodology applicable across domains. The semantic web research effort has produced languages like DAML+OIL (an extension to XML and RDF) [DAML] as a means of communication among disparate entities. It adds meaning to RDF triples, which talk about objects and relationships. DAML-Space and DAML-Time are ontologies for specification of spatial and temporal characteristics, respectively. DAML+OIL, OWL [OWL2004], FOAF [Dumbill2002] and SOUPA [Chen2004] are all proposed ontologies that are relevant to ubicomp research. SOUPA was proposed by the designers of Rei, and has support for certain *core* features like entities, agents, events, actions, policies (including deontic concepts) and contextual parameters like space and time. *Extensions* can be defined in an application or domain-specific manner. SOUPA is promising research, and will be watched closely and leveraged in my policy language.

7.3. Ubiquitous Interoperation and Service Discovery

Here we examine how the ubiquitous computing projects targeting smart spaces handle interoperation in an automated manner, and how these and other open systems handle the core problems of service discovery, resource management and access control.

The most prominent smart space projects typically look at different components of an active space (computing entities, resources, physical interfaces) as parts of a whole, rather than independent entities in their own right. Therefore, each of these projects manages all the entities within in a centralized manner. Metaglue [Coen1999] is an extension of Java that provides the computational glue for interoperation of software agents in an Intelligent Room [Adjie-Winoto1999][Brooks1997], which is a product of MIT's Oxygen [MIT-Oxygen] project. Gaia [Román2002] and One.world [Grimm2004a][Grimm2004b] are the equivalent of operating systems for pervasive computing systems, which manage resources, devices and applications within a domain. The centralized management works well within a limited domain, but does not scale. These systems also did not take the security aspect seriously, at least at the beginning, and do not handle unknown and un-trusted devices very well. What these systems do very well is to facilitate ad hoc interactions in a seamless manner, manage resource (or agent) discovery and allocation in a dynamic manner, and adapt to a limited range of context changes. The price that needs to be paid for such seamless operation is standardization of hardware and software components and application designs. Interoperation within a room is limited to familiar devices that know what to expect and have ways of obtaining those resources. They also do not consider the management of multiple active spaces. One.world, as an example, provides flexibility through an application-oriented approach that assumes the trustworthiness of devices, in contrast to my device-centric approach.

Lately, these systems have been augmented with security features. Hyperglue was designed to enable interactions among multiple active spaces and remove the centralized management [Kottahachchi2004][Peters2003]. Still, interactions are managed by a DNS-style lookup with questionable scaling properties. Hyperglue has the nice property of letting domains manage themselves independently and interacting with others as a single virtual entity. It also

performs access control using an RBAC scheme. Even though permission granting is context-variant, the assignment of roles is limited to known entities or entities that can prove transitive relationships. With a limited trust model and the lack of a flexible negotiation scheme, the interoperation features provided fall short of ubicomp requirements. Cerberus [Al-Muhtadi2003], a Gaia security extension, uses policies and *confidence levels* in authentication schemes to enforce access control in a context-aware and non-intrusive manner. It also suffers from the same drawback as Hyperglue, as it uses security policies simply to validate access, and not for dynamic service discovery and negotiation, as I plan to do in my research. Mobile Gaia [Chetan2004] extends the basic Gaia design to manage ad hoc clusters, like Panoply spheres, but the negotiation is limited to the production of a familiar public key that can be authenticated by a cluster. Gaia Super Spaces [Al-Muhtadi2004] handles multiple active domains in a semi-centralized manner, interoperations occurring through *bridges*. Super Spaces provides service lookup and access functions across domains, though without considering the security aspects. Centaurus 2 [Undercoffer2003] and the Aware Home project [Kidd1999] are yet more examples of ubiquitous active spaces systems, though these place a high priority on security. The results are not very different from Hyperglue or Cerberus, however. Centaurus 2 enables secure interaction within a hierarchy of spaces through access capabilities, and an Aware Home uses the powerful GRBAC model for access control. Still, the security aspect is independent of the service lookup and management module, and security policies are enforced in a way so that only known entities with predeployed trust information may obtain access permissions for services.

Jini [Waldo1999], a Java-based technology, enables autonomous service discovery and resource access over a network connection. Devices can register, discover services and access them through proxies, lookup tables and leasing mechanisms. Standard interfaces and mobile code enable spontaneous interoperation, since every device in a Jini-enabled space speaks Java and can communicate through RMI, which is a fairly ubiquitous property. Jini is easier to use and maintain than similar frameworks like CORBA or DCOM, where protocol changes must be synchronized among servers and clients offline. Jini does its primary job of service discovery and hookup very well, though the concept of open interfaces is completely unacceptable where there are even minimal security and privacy concerns. The model by itself, even with authentication and authorization mechanisms, works in a static domain that serves a set of known client devices and does not adapt to context change. Dynamic policy-guided negotiation expands the scope of the interoperation problem and solves complementary issues while retaining the useful features of Jini. *Universal Plug and Play* [UPnP] is another communication architecture based on well-known technologies like TCP/IP, HTML and XML that allows seamless, spontaneous networking among suitably configured devices irrespective of hardware or operating system characteristics. Devices can advertise their capabilities and learn about other devices' capabilities through SSDP, GENA and SOAP protocols. In the context of my research aims, the contributions stop here. Security is handled by using ACLs and certificates which the users are expected to maintain in a static manner; this is non-scalable and intrusive, and therefore not suitable enough for ubicomp.

Zhu and others [Zhu2005] outline a service discovery protocol for pervasive computing in one of the few systems where the security aspects of discovery are considered. In the absence of a trusted third party, service provider and client expose partial sensitive information in a progressive approach till both parties reach an agreement about exposure of the nature of the service and authentication information respectively. Upon a mismatch or an unsatisfied request, the protocol can be terminated without loss of privacy. The key drawback from my point of view is that the entities are assumed to share security information, and the protocol is essentially a way of preventing malicious devices from causing privacy violations. These constraints cannot be assumed in the more general negotiation problem that I am addressing.

7.4. Models for Trust and Access Control

In this prospectus I have discussed why access control (especially when mixed with trust issues) merges with service discovery for security and privacy reasons in open systems providing ubiquitous services. Negotiation, therefore, is a different way of handling access control, though it uses some of the existing concepts and designs. Here we look at currently used models for access control and trust building and observe their drawbacks when applied to ubicomp interoperation. Individual domains will use trust and access control models to frame policies and negotiation heuristics; from this point of view, we can consider these models to be complementary research.

ACLs (access control lists) and capabilities are basic mechanisms for enforcing access control when familiar entities access known objects, and still retain their use in limited domains where one can set per-entity and per-object policy. Policy expressivity is very limited and rules are rigidly enforced. These mechanisms suffer from scalability issues when applied directly to ubicomp scenarios, and also don't adjust to changing context. They can be used in individual domains in a ubiquitous computing environment, but each of these domains must have a security and trust management framework in addition. Frameworks like Kerberos enforce access control through secure protocols, though they also do not scale beyond a small self-contained domain.

Most open systems, and those that need to handle access control for a large number of entities, typically use RBAC (role-based access control) [Ferraiolo1995] models or its variations. Entities are assigned *roles*, which are associated with sets of access privileges, in the most basic model. Though it scales better than ACLs, it does not reach the flexibility level desired in ubicomp. Since roles have to be defined in advance, it is difficult to provide optimal security in dynamic situations where policies need to be adjusted with context. Generalized RBAC [Covington2000] enhances RBAC by adding roles for accessible objects and environment states (context), and has been used in the Aware Home ubiquitous computing project [Kidd1999]. It increases the expressivity and intuitiveness of policy writing, and as such can be used in negotiation trust models. Delegation of permissions is an extremely useful concept, and has impressive scaling properties, though it has trust issues and requires something like our negotiation model before it gains widespread adoption. The distributed RBAC [Freudenthal2002] model is another variation of RBAC, which uses delegation and proof building of permission rights using knowledge of possessed credentials. I will leverage concepts proposed by GRBAC and DRBAC in my negotiation model.

Drawbacks of pure RBAC when used in open systems have been addressed in other recent systems. *Law governed interaction* [Minsky2000] has the view, which we share, that role semantics should be dictated by context and policy, the latter being independent of particular role definitions. This helps to rectify drawbacks inherent in RBAC, such as difficulty in specification of exception conditions, and the potential security holes that could result. The key drawback with LGI is that it assumes a common *law* or policy governing all interacting domains, which is impractical, as I have argued earlier when discussing the need for negotiation. Another access control model that specifically targets pervasive computing environments [Hengartner2003][Hengartner2005] uses information relationships and per-domain policy specifications to ensure that entities release minimum sensitive information. If multiple entities are involved in the transfer of such information, each entity gets information on a need-to-know basis. We tackle an orthogonal, though overlapping problem of unfamiliar entities having to negotiate, not only for access to services, but to discover those services in the first place.

Trust has been studied in recent years, both in the context of theoretical models and for practical use in open systems. It has always been used in limited ways in computer security, based either on identity or trusted authorities. It would be fair to say that the research community generally accepts the notion of trust as a basis for secure interactions in ubicomp [English2002].

PolicyMaker [Blaze1998] and KeyNote [Blaze1999] were seminal trust management projects where credentials (typically public keys) were tied to the permissions they represented rather than identity. An access required the production of a key which would be input to a compliance checker along with a request and a policy, the output being a yes/no answer. It was a simple and powerful model, but it has been overtaken by more sophisticated schemes that handle a much wider range of situations than Policy Maker was envisioned for.

The Secure project [English2002][Cahill2003] argues for a dynamic notion of trust, with a history of past interactions being used as a basis for trust formation, and additional evidence leading to trust evolution. This trust information is then used for access control decisions using appropriately framed security policies. Secure also presents formal models that relate trust to events and results, and trust value changes based on evidence. Trust building through reputation frameworks, or collection of evidence from known sources, has been well studied [Xiong2004]. Such systems have limited use in practical security, mainly because of the huge number of variables involved and the possibility of entities lying and colluding [Sen2002]. Formal reputation-based trust models generally use probabilistic reasoning, and do not clearly specify the decision-making processes. In this context, models such as a calculus for access control [Abadi1993] and formal trust dynamics using temporal logic [Marx2001] bear looking into.

7.5. Other Research

The **Semantic Web** is work in progress toward a common framework that allows data to be shared and reused across application, enterprise, and community boundaries [SemWeb]. Still, certain standards have been established, notably the Resource Discovery Format [RDF], which is the semantic data description model that uses XML syntax and URI naming procedures in order to allow applications to interoperate. The complete scope of the semantic web is not very clear, though it certainly includes cross-platform description of data and resources, the prime achievement of the project thus far. Some researchers also consider the definition of pervasive computing ontology and security policies to be within its ambit [Kagal2003b], which would place my research in policy-guided negotiation within its overall scope. Whatever the scope, cross-platform and cross-application description of data and resources is complementary to the process of negotiation, as different domains and devices need to understand what they are talking about before they can negotiate. To sum up, if the semantic web described the language of ubicomp interoperation, my research describes the mechanics.

P3P (Platform for Privacy Preferences) [P3P] is an intended standard for management of user privacy on the web. This allows websites to describe their privacy policies in a standardized format (currently in XML). This information can be used by web clients on user devices, which have user policies specified (also in XML), to make judgments on whether or not private user information can be released to the remote sites. Feedback on policy conflicts is given to the user, who can then make an information release decision. P3P suffers from many limitations, such as the lack of a model for website trust and verification of policies, and an expressive policy language, leading to low adoption [Kolari2005][Kagal2003b]. Its use is limited to information that websites want to obtain, and as such cannot be used for general information privacy, much less for full-fledged ubicomp interaction. The reasoning framework is limited to simple matching with no conflict resolution, and web servers have a rigid policy that precludes negotiation. Certain enhancements have been proposed, such as using the more expressive Rei language, trust management using reputation frameworks, and allowing users to set per-website and data item preferences. Still, P3P will leave too much to users and depend on rigid adherence to policies, requiring significant additional research to make it suitable for ubicomp negotiation.

8. Conclusion

A decentralized negotiation procedure between devices and domains is the best way to interoperate spontaneously in a ubiquitous computing environment. This research assumes the presence of basic infrastructure for data communication, and leverages semantic web research for common application layer understanding of objects and policies. It does not assume the presence of any kind of established identity relationships or any trust infrastructure. Local policies are assumed to be private. Negotiation itself is based on trust and utility models, and different entities will adopt different strategies based on risk-benefit analysis computed using these models, along with policy and contextual information. Policies will be easy to specify at a high level by ordinary users and will be interpreted by the negotiation framework based on context. The key contribution of this research is a much more flexible procedure of interactions between computers and agents, where an ideal agreement can be reached somewhere in the spectrum between a completely open system and a system that enforces rigid, invariant policy. It also contributes toward making computing devices more autonomous and less dependent on user feedback. Negotiation capability added to *spheres of influence* makes the Panoply middleware a powerful model for ubiquitous computing.

References

[**Abadi1993**] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon D. Plotkin, "A Calculus for Access Control in Distributed Systems," *ACM Transactions on Programming Languages and Systems*, 15(4):706--734, September 1993**.**

[**Adjie-Winoto1999**] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan and J. Lilley, "The Design and Implementation of an Intentional Naming System." *ACM SIGOPS Operating Systems Review, v.33 n.5*, p.186-201, Dec. 1999.

[**Al-Muhtadi2003**] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. Mickunas, "Cerberus: A Context-Aware Security Scheme for Smart Spaces," *First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, March 23-26, 2003.

[**Al-Muhtadi2004**] Jalal Al-Muhtadi, Shiva Chetan, Anand Ranganathan and Roy Campbell, "SuperSpaces: A Middleware for Large-Scale Pervasive Computing Environments", *Perware '04: IEEE International Workshop on Pervasive Computing and Communications*, Orlando, Florida, March 2004.

[**Anius2003**] Diana Anius, "Enhancing Innovation within a Regional Wireless Grid," *International Conference on Computer, Communication and Control Technologies (CCCT '03)*, Orlando, Florida, July 31-August 2, 2003.

[**Bagrodia2003**] R. Bagrodia, S. Bhattacharyya, F. Cheng, S. Gerding, G. Glazer, R. Guy, Z. Ji, J. Lin, T. Phan, E. Skow, M. Varshney and G. Zorpas, "iMASH: Interactive Mobile Application Session Handoff," *In Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, May 2003.

[**Bertino2001**] E. Bertino, S. Castano and E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-Based Language," *In Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 57--65, Chantilly, VA, May 2001. ACM Press.

[**Bhagwat1996**] P. Bhagwat, C. Perkins and S. Tripathi, "Network Layer Mobility: An Architecture and Survey," *Personal Communications, IEEE*, 3:54--64, June 1996.

[**Blaze1998**] M. Blaze, J. Feigenbaum and M. Strauss, "Compliance Checking in the PolicyMaker TrustManagement System," *In Proceedings of the Financial Cryptography Conference, Lecture Notes in Computer Science*, vol. 1465, pages 254--274. Springer, 1998.

[**Blaze1999**] M. Blaze, J. Feigenbaum, J. Ioannidis and A. D. Keromytis, "The KeyNote Trust Management System Version 2," *RFC 2704* (September 1999).

[**Bonatti2000**] P. Bonatti and P. Samarati, "Regulating Service Access and Information Release on the Web," *In Conference on Computer and Communications Security*, Athens, Nov. 2000.

[**Brooks1997**] R. Brooks, "The Intelligent Room Project," *Proceedings of the 2nd International Cognitive Technology Conference*. 1997. Aizu, Japan.
[**Bruneo2003**] Dario Bruneo, Marco Scarpa, Angelo Zaia and Antonio Puliafito, "Communication Paradigms for Mobile Grid Users," *3rd International Symposium on Cluster Computing and the Grid*, 2003, *ccgrid*, p. 669.
[**Burrows1990**] M. Burrows, M. Abadi and R. Needham, "A Logic of Authentication," *ACM Trans. Computer Systems* 8(1), 1990, 18-36.
[**Cahill2003**] Vinny Cahill, Elizabeth Gray, Jean-Marc Seigneur, Christian D. Jensen, Yong Chen, Brian Shand, Nathan Dimmock, Andy Twigg, Jean Bacon, Colin English, Waleed Wagealla, Sotirios Terzis, Paddy Nixon, Giovanna di Marzo Serugendo, Ciaran Bryce, Marco Carbone, Karl Krukow and Mogens Nielsen, "Using Trust for Secure Collaboration in Uncertain Environments," *IEEE Pervasive Computing*, vol. 02, no. 3, pp. 52-61, July-September, 2003.
[**Chao2002**] K-M Chao, R. Anane, J-H. Chen and R. Gatward, "Negotiating Agents in a Market-Oriented Grid," *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002, *ccgrid*, p. 436.
[**Chen2004**] Harry Chen, Filip Perich, Timothy W. Finin and Anupam Joshi, "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications," *MobiQuitous 2004*: 258-267.
[**Chetan2004**] Shiva Chetan, Jalal Al-Muhtadi, Roy Campbell and M.Dennis Mickunas, "A Middleware for Enabling Personal Ubiquitous Spaces", *UbiSys '04: System Support for Ubiquitous Computing Workshop at Sixth Annual Conference on Ubiquitous Computing (UbiComp 2004)*, Nottingham, England, Sept. 2004.
[**Coen1999**] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters and Peter Finin, "Meeting the Computational Needs of Intelligent Environments: The Metaglue System," *In 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, pp.201--212. Dublin, Ireland, December 1999.
[**Covington2000**] Michael J. Covington, Matthew J. Moyer and Mustaque Ahamad, "Generalized Role-Based Access Control for Securing Future Applications," *Technical Report GIT-CC-00-02, Georgia Institute of Technology, College of Computing*, February 1, 2000.
[**Czajkowski2002**] Karl Czajkowski, Iand Foster, Carl Kesselman, Von Sander and Steven Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," *In 8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.
[**Damianou2001**] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language," *Policy Workshop 2001*, Jan. 2001, Bristol, U.K.
[**DAML**] "The Darpa Agent Markup Language Homepage," http://www.daml.org
[**Dumbill2002**] Ed Dumbill, "XML Watch: Finding friends with XML and RDF," *IBM Developer Works*, http://www-106.ibm.com/developerworks/xml/library/x-foaf.html, June 2002.
[**English2002**] C. English, P. Nixon, S. Terzis, A. McGettrick and H. Lowe, "Dynamic Trust Models for Ubiquitous Computing Environments," *In Proceedings of Workshop on Security in Ubiquitous Computing, Ubicomp 2002*, 2002.
[**Eustice2003a**] Kevin Eustice, Leonard Kleinrock, Shane Markstrum, Gerald Popek, V. Ramakrishna and Peter Reiher, "Enabling Secure Ubiquitous Interactions," In the proceedings of the *1st International Workshop on Middleware for Pervasive and Ad-Hoc Computing (in conjunction with Middleware 2003)*, 17 June 2003 in Rio de Janeiro, Brazil.
[**Eustice2003b**] K. Eustice, L. Kleinrock, S. Markstrum, G. Popek, V. Ramakrishna and P. Reiher, "Securing WiFi Nomads: The Case for Quarantine, Examination, and Decontamination," *Proceedings of the New Security Paradigms Workshop (NSPW) 2003*.
[**Ferraiolo1995**] D. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role Based Access Control (RBAC): Features and Motivations," *Proc. 1995 Computer Security Applications Conference*, December 1995, p241-248.
[**Ferreria2002**] Vincent Ferreria, Alexey Rudenko, Kevin Eustice, Richard Guy, V. Ramakrishna and Peter Reiher, "Panda: Middleware to Provide the Benefits of Active Networks to Legacy Applications," *DANCE 02*, May 2002.
[**Fishburn1988**] P. C. Fishburn, "Nonlinear Preferences and Utility Theory," *John Hopkins University Press, Baltimore*, 1988.
[**Freudenthal2002**] E. Freudenthal, T. Pesin, L. Port, E. Keenan and V. Karamcheti, "dRBAC: Distributed Role-Based Access Control for Dynamic Coalition Environments," *In Proceedings of the 22nd*

International Conference on Distributed Computing Systems (ICDCS'02,. IEEE Computer Society, July 2002.
[**Gavriloaie2004**] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons and M. Winslett, "No Registration Needed: How to Use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web," *In Proceedings of the 1st First European Semantic Web Symposium*, Heraklion, Greece, May 2004.
[**Google2005**] "Google proposes free Wi-Fi for San Francisco", http://news.yahoo.com/s/nm/20051001/wr_nm/google_wifi_dc.
[**GRID**] "Grid Computing Info Centre (GRID Infoware)," http://www.gridcomputing.com/.
[**Grimm2004a**] Robert Grimm, "One.world: Experiences with a Pervasive Computing Architecture". *IEEE Pervasive Computing*, 3(3):22-30, July-September 2004.
[**Grimm2004b**] Robert Grimm, Janet Davis, Eric Lemar, Adam MacBeth, Steven Swanson, Thomas Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble and David Wetherall, "System Support for Pervasive Applications" (PDF, 1,777 KB). *ACM Transactions on Computer Systems*, 22(4):421-486, November 2004.
[**Guttman2001**] Erik Guttman, "Autoconfiguration for IP Networking: Enabling Local Communication," *IEEE Internet Computing*, v.5 n.3, p.81-86, May 2001.
[**Hengartner2003**] U. Hengartner and P. Steenkiste, "Access Control to Information in Pervasive Computing Environments," *Proc. of 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, Lihue, HI, May 2003, pp. 157-162.
[**Hengartner2005**] U. Hengartner and P. Steenkiste, "Exploiting Information Relationships for Access Control," *Proc. of Third IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, Kauai Island, HI, March 2005, pp. 269-278.
[**Herzberg2000**] A. Herzberg, Y. Mass, L. Mihaeli, D. Naor and Y. Ravid, "Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers," *In Proc. Symposium on Security and Privacy*, pages 2--14, 2000.
[**IEEE802.16**] "IEEE 802.16 Backgrounder", http://grouper.ieee.org/groups/802/16/pub/backgrounder.html
[**Kagal2001a**] L. Kagal, V. Korolev, H. Chen, A. Joshi and T. Finin, "Centaurus: A Framework for Intelligent Services in a Mobile Environment", *21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*, April 16 - 19, 2001, Mesa, Arizona.
[**Kagal2001b**] L. Kagal, T. Finin and A. Joshi, "Moving from Security to Distributed Trust in Ubiquitous Computing Environments", *IEEE Computer*, December 2001.
[**Kagal2003a**] L. Kagal, T. Finin and A. Joshi, "A Policy Language for a Pervasive Computing Environment," *In IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003.
[**Kagal2003b**] Lalana Kagal, Tim Finin and Anupam Joshi, "A Policy Based Approach to Security for the Semantic Web," *InProceedings, 2nd International Semantic Web Conference (ISWC2003)*, September 2003.
[**Kidd1999**] C. D. Kidd, R. J. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner and W. Newstetter, "The Aware Home: A Living Laboratory for Ubiquitous Computing Research," *In the Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99*, October 1999.
[**Kindberg2002**] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing, vol. 1, no. 1*, Jan.-Mar. 2002, pp. 70-81.
[**Kolari2005**] Pranam Kolari, Li Ding, Shashidhara Ganjugunte, Anupam Joshi, Timothy W. Finin and Lalana Kagal, "Enhancing Web Privacy Protection through Declarative Policies," *POLICY 2005*: 57-66.
[**Kottahachchi2004**] Buddhika Kottahachchi and Robert Laddaga, "Building Access Controls for Intelligent Environments," *In Proceedings of ISDA '04: 4th Annual International Conference on Intelligent Systems Design and Applications*. Budapest, Hungary, August 2004.
[**Lawley2003**] Richard Lawley, Keith Decker, Michael Luck, Terry R. Payne and Luc Moreau, "Automated Negotiation for Grid Notification Services," *Euro-Par 2003*, 384-393.
[**Lorch2003**] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura and Sumit Shah, "First Experiences Using XACML for Access Control in Distributed Systems," *Proceedings of the 2003 ACM workshop on XML security*, October 31-31, 2003, Fairfax, Virginia.
[**Marx2001**] M. Marx and J. Treur, "Trust Dynamics Formalised in Temporal Logic," In: L. Chen, Y. Zhuo(eds.), *Proceedings of the Third International Conference on Cognitive Science*, ICCS 2001. USTC Press, Beijing, pp. 359-363.

[**Minsky2000**] Naftaly H. Minsky and Victoria Ungureanu, "Law-governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, v.9 n.3, p.273-305, July 2000.
[**MIT-Oxygen**] "MIT Project Oxygen: Overview", http://www.oxygen.lcs.mit.edu/Overview.html
[**Nejdl2004**] Wolfgang Nejdl, Daniel Olmedilla and Marianne Winslett, "PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web," *Secure Data Management 2004*, 118-132.
[**Owen1995**] Guillermo Owen, "Game Theory," *Academic Press*, 1995.
[**OWL2004**] "OWL Web Ontology Language Overview," http://www.w3.org/TR/owl-features/
[**P3P**] "P3P Public Overview": http://www.w3.org/P3P/.
[**Peters2003**] S. Peters, G. Look, K. Quigley, H. Shrobe and K. Gajos, "Hyperglue: Designing High-Level Agent Communication for Distributed Applications," *Originally submitted to AAMAS'03*.
[**RDF**] "Resource Description Framework (RDF) / Semantic Web Activity": http://www.w3.org/RDF/.
[**Román2002**] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pp. 74-83, Oct-Dec 2002.
[**Roy1990**] Peter Van Roy, "Can Logic Programming Execute as Fast as Imperative Programming?," *PhD thesis, University of California at Berkeley*, November 1990.
[**Schaad01**] A. Schaad. "Detecting Conflicts in a Role-Based Delegation Model," *Proceedings of the 17th Annual Computer Security Applications Conference*, vol. 00, no. , p. 0117, 17th 2001.
[**Seamons2002**] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills and L. Yu, "Requirements for Policy Languages for Trust Negotiation," *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, Monterey, CA, June 2002.
[**SemWeb**] "W3C Semantic Web home page": http://www.w3.org/2001/sw/.
[**Sen2002**] Sandip Sen and Neelima Sajja, "Robustness of Reputation-Based Ttrust: Boolean Case," *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*: part 1, July 15-19, 2002, Bologna, Italy.
[**SWIProlog**] "SWI-Prolog's Home," http://www.swi-prolog.org
[**Undercoffer2003**] J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal and A. Joshi, "A Secure Infrastructure for Service Discovery and Access in Pervasive Computing," Article, *ACM Monet: Special Issue on Security in Mobile Computing Environments*, October 2003.
[**UPnP**] "UPnP Forum," http://www.upnp.org
[**Waldo1999**] J. Waldo, "The Jini Architecture for Network-Centric Computing," *Communications of the ACM*, Vol. 42, No. 7, p.76-82, 1999.
[**Weiser1991**] M. Weiser, "The Computer for the 21st Century," *Scientific American 265(30)*, pp. 94-104, 1991.
[**Winsborough2000**] W. H. Winsborough, K. E. Seamons and V. E. Jones, "Automated Trust Negotiation," *DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, January 2000.
[**Winslett2002**] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith and L. Yu, "Negotiating Trust on the Web," *IEEE Internet Computing*, November/December 2002.
[**Winslett2003**] M. Winslett, "An Introduction to Trust Negotiation," *1st International Conference on Trust Management*, Crete, Greece, May 2003.
[**Xiong2004**] L. Xiong and L. Liu, "PeerTrust: Supporting Reputation-Based Trust in Peer-to-Peer Electronic Communities," *IEEE Transactions on Knowledge and Data Engineering (TKDE), Special Issue on Peer-to-Peer Based Data Management*, 2004.
[**Yarvis1999**] M. Yarvis, A. A. Wang, A. Rudenko, P. Reiher and G. J. Popek, "Conductor: Distributed Adaptation for Complex Networks," *Technical Report CSD-TR-990042*, University of California, Los Angeles, Los Angeles, California, August 1999.
[**Yu2001**] T. Yu, M. Winslett and K. E. Seamons, "Interoperable Strategies in Automated Trust Negotiation," *8th ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, November 2001.
[**Zhu2005**] Feng Zhu, Wei Zhu, Matt W. Mutka and Lionel M. Ni, "Expose or Not? A Progressive Exposure Approach for Service Discovery in Pervasive Computing Environments," *PerCom 2005*: 225-234.