Putting Computing in Context: An Infrastructure to Support Extensible Context-Enhanced Collaborative Applications

W. KEITH EDWARDS
Georgia Institute of Technology

Context-aware computing exposes a unique tension in how information about human context is modeled and used. On the one hand, approaches that use loosely structured information have been shown to be useful in situations where humans are the final consumers of contextual information; these approaches have found favor in many CSCW applications. On the other hand, more rigidly structured information supports machine interpretation and exchange; these approaches have been explored in the ubiquitous computing community. The system presented here, dubbed Intermezzo, represents an exploration of a space between these two extremes. Intermezzo combines a loose data structuring with a number of unique features designed to allow applications to embed specialized semantic interpretations of data into the infrastructure, allowing them to be reused by other applications. This approach can enable the construction of applications that can take advantage of rich, layered interpretations of context without requiring that they understand all aspects of that context. This approach is explored through the creation of two higher-level services that provide context-enhanced session management and context-enhanced access control.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Collaborative computing*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems, current awareness systems*

General Terms: Design, Human Factors

Additional Key Words and Phrases: Collaboration, context-aware computing, data representations, coordination

1. INTRODUCTION

All work—like everything else in our lives—occurs in some setting. And although we typically think of the setting of work as the place where that work occurs, setting is far broader than just physical location. Setting includes the people around us or collaborating with us; the work of other people with whom we may interact; the reasons, demands, and often unspoken constraints

Author's address: College of Computing, Georgia Institute of Technology, GVU Center, 85 Fifth Street NW, Atlanta, GA 30332-0760; email: keith@cc.gatech.edu.

surrounding the tasks at hand. In our increasingly computer-mediated lives, it also involves the virtual aspects of work such as the applications we use, our online coworkers, files, and so on. In short, the setting of work is more appropriately defined as the entire context in which the work occurs—the web of people, events, places, and activities that are both explicit and implicit in the work itself.

Because context is such an essential part of how we interact, there is great potential, and hence interest, in so-called *context-aware computing*: computational support for the sharing and use of human contextual information.

And yet, despite the recent focus on it (particularly in the area of ubiquitous computing), context-aware computing is not a new research area. In particular, the computer-supported cooperative work (CSCW) literature has a long history of focus on tools that represent, transmit, and store information about people and their situations and how this information can be used to help users coordinate with each other. While many of these tools are purely media-oriented (e.g., Cruiser [Root 1988] and Mediaspace [Bly et al. 1993]), using video and audio to allow users to maintain general awareness of the context of colleagues, others support the sharing of context through domain-specific tools (e.g., ShrEdit [McGuffin and Olsen 1992], PREP [Neuwirth et al. 1990]) and techniques such as Radar Views [Gutwin et al. 1996], all of which convey a user's context within a particular application. In these systems, even though applications mediate the presentation of context, it is ultimately intended for consumption by humans. In these applications, the contextual information that is shared is about people and intended for use by people.

This focus on people as the consumers of contextual information is in contrast to the focus of much of the work in the ubiquitous computing community. In that work, *systems* rather than *people* are often assumed to be the ultimate consumers of contextual information about users. For example, tools such as CyberGuide [Abowd et al. 1997] and the Conference Assistant [Dey et al. 1999] and others use location data that is parsed and understood by the application in order to adapt its behavior to its situation of use.

These two contrasting uses for information about people lead to different infrastructure mechanisms to support its representation, sharing, and use. In CSCW systems, contextual information tends to be produced and consumed by multiple instances of a single application. In such circumstances, an application developer can decide on the format for representing context without consideration for how it might be used by other applications. In ubiquitous computing systems, on the other hand, context is typically generated through some "capture infrastructure" such as the Context Toolkit [Salber et al. 1999] and then made available to other applications. In such a situation, the format of the information must be agreed upon by the capture infrastructure that produces it as well as by the applications that must understand the syntax and semantics of it in order to act appropriately.

There are representational differences in addition to usage differences. In many CSCW systems, the format of context information is only loosely structured since it will often be consumed by humans. For example, the Elvin tickertape application [Fitzpatrick et al. 1998] exchanges unstructured strings.

Because the application relies on humans to parse and understand the transferred data, there is no need for more formal structuring. This is especially true since the data is often shared only within multiple instances of a single application, and thus can be easily extended or augmented without fear of breaking other applications.

These different approaches lead, in turn, to a fundamental trade-off that constrains how applications make use of context. Obviously, unstructured and even loosely structured data can be difficult for systems to parse. But on the other hand, the very formal structure that lends itself to machine parsing can lead to inflexibility and brittleness. For example, any given representation will define a syntax that encodes a set of context features that may implicitly presume an intended use (by representing certain features of a person's context and not others, by encoding attributes like location to a certain precision, etc.). New applications which may bring new requirements can be difficult to retrofit on top of the representation, leading to the need to evolve its format. These problems are compounded when the information is meant to be shared across applications. Because these applications depend on agreement about the syntax and semantics of the context, the representations often cannot easily evolve to accommodate new semantics without breaking compatibility.

While this trade-off between structured and unstructured representations exists for many types of information, it is especially problematic in the case of context. First, context represents information about people that is very often ambiguous by nature, subtle in its interpretation, and can be applied to many uses. Second, there is a great range of information about humans that is potentially useful (ranging from general information about users' locations or actions, to domain-dependent information such as a user's context in a specific application). Third, different sorts of context are important to different applications. For example, physical copresence may be salient for a single display groupware application [Stewart et al. 1999], while the fact that multiple users are editing the same document may be salient for a shared editing tool.

Put simply, while highly-structured data representations are amenable to use by applications (they can be easily machine parsed, processed, and stored), they are problematic in situations where the needs of applications are evolving, where the range of information that must be represented is very great, and when agreement among multiple applications is required, in other words, the very situations posed by context-aware computing.

The traits of context—the richness and range of it, and the breadth of needs of applications that will use it—make it difficult to define a formal, overarching ontological structure that defines all of the relevant aspects of context. Paradoxically however, without such an ontology, we may lack the ability for systems to process the information and especially the ability to share it between applications.

This work takes the approach that, because the use and meaning of context is evolving, fluid, and ambiguous, it is impractical for infrastructure designers to try to predict all of the aspects of context that will be meaningful to applications or to people. In other words, we cannot yet define an ontology of context that will serve all applications for all uses and without such an ontology, we must be prepared to deal with multiple interpretations, extensible representations, ambiguity, and evolution. The approach taken by the infrastructure described in this article, dubbed *Intermezzo*, is to provide representations that support these needs by *allowing applications to impose their own semantic constraints on the data store* in such a way that these constraints can not only evolve, but also support at least some degree of reuse by other applications. The primary contribution of this work is in its exploration of this data model for exploiting context in collaborative systems.

The rest of this article is structured as follows. The next section explores previous work in the area of infrastructures to support context aware applications both from the CSCW and ubiquitous computing fields. This section also explores in more detail the representational tensions that arise when dealing with context as a way of more fully motivating the approach taken by Intermezzo. After this, the next section explores the data model used by Intermezzo to represent context. This model combines features of simple object databases with a notification service and adds some features particularly intended to support the extensible modeling and sharing of context. Then the article discusses how the Intermezzo data model can be used to support the creation of higher-level building blocks that can be used to facilitate context-enhanced collaborative applications. Two of these building blocks are explored, a context-enhanced session management service and a facility for context-enhanced access control. Both of these use the underlying Intermezzo platform to support cross-application coordination through sharing extensible representations of the context of their users. The article discusses Intermezzo from the perspective of application writers, and concludes with a summary and an exploration of the lessons learned from leveraging powerful representations as a tool to support coordination.

2. MOTIVATION AND RELATED WORK

As noted earlier, a number of systems have addressed the goal of supporting applications that are responsive to the states and changes of the context of their users. In the CSCW community, many of these tools have focused on providing coordination support to collaborative applications and share certain requirements in common (including, as Ramduny et al. [1998] note, the ability to access and update shared data and to be notified when that application data has updated). These requirements have been fulfilled by various systems, including *notification services*. Notification services are publish-subscribe infrastructures that allow applications to publish data items to zero or more subscribers which are applications that have registered to receive such notifications. These systems typically allow coordination among loosely coupled parties in which a publisher may neither know nor care whether any subscribers are listening for updates.

There have been a number of these notification services constructed, some with CSCW applications specifically in mind and some not. For example, the Elvin system [Fitzpatrick et al. 1999] is a well-known notification service that has been applied successfully to a number of collaborative applications. Elvin is, in the terms established by Ramduny et al. [1998] a pure notification service, that is, it performs notifications only (it is not also a shared data store), and propagates those updates asynchronously to clients that have solicited interest in them. The Lotus PlaceHolder system [Day et al. 1997] is another notification service. PlaceHolder, however, also incorporates a persistent data store that can be used by applications to manage shared data; changes to this data by any client result in notifications that are generated to other interested clients. Thus, in the terms of Ramduny et al., PlaceHolder implements the gatekeeper pattern in which the notification service is tightly bound with an integrated data store.

For the most part, all of these notification services have very simple data models. PlaceHolder, for instance, stores *things* which are simply named byte arrays that applications are meant to interpret according to their particular semantics (in other words, for applications to share data through PlaceHolder, they must agree not only on the naming conventions for things, but also on the format of the values stored in them, and the semantics of that data). Although it does not persistently store data, Elvin has a similar data model that it uses to describe the content of events and event subscriptions, named attributes comprising a variety of simple data types. These systems are typical of the loosely structured representations that have been adopted by the CSCW community.

A number of data repositories have models very similar to these notification services. For example, tuplespace systems such as Gelernter's Linda [1985], the Stanford EventHeap [Johanson et al. 2002], and JavaSpaces [Arnold et al. 2999], allow applications to store untyped tuples (collections of named data elements) in the model. Likewise, Placeless Documents [Dourish et al. 2000] presents a loosely-structured data model, named properties that contain unstructured data, similar to that of both Elvin and PlaceHolder.

Such arrangements yield great power. They allow the infrastructure to be very simple because the data model is highly generic—the infrastructure need not know the semantics or structure of the data it stores—and can also be applied to a wide range of applications. They are also highly adaptable in that new applications can use existing data in the model, and add their own without breaking the infrastructure. The expressiveness of these approaches facilitates such easy adaptability.

There is, however, a downside to this semantic-free representation. Because the sharing infrastructure is generic—meaning that it imposes little structure and virtually no meaning on the data it stores—applications are left to their own devices to create higher level structure and impose semantics on the data they share. Since these structures often emerge on an ad hoc, application-by-application basis (and since the infrastructure doesn't understand and enforce their syntax and semantics), they exacerbate the problems of agreement among clients on how shared data will be represented and intepreted. This limits the cross-application synergy and reuse that is crucial to coordination: if we expect applications to be able to share a data model, they must agree on the terms in which that data will be represented, the semantics of it, and so on.

In situations where the data is very simple (as in the case of the Elvin tickertape [Fitzpatrick et al. 1998]), where it is designed for human consumption (as in the case of Gutwin et al.'s awareness widgets [1996]), or when only one or a small handful of clients will access it (for example, a set of bespoke tools explicitly designed to work together, or multiple instances of a single application), these problems are mitigated.

On the other hand, these problems are accentuated when the data that is shared is intended for machine (as opposed to human) interpretation or if it's very complex, especially if we expect multiple clients to use it. These are the very sorts of conditions that are important for cross-application coordination.

A common approach that sacrifices ad hoc evolution in favor of rigid cross-application agreement is to create ontologies that specify the name and value spaces of the data. Such ontologies represent an attempt to standardize the names, formats, and semantics of data at a fine-grained level; effectively, they preordain all aspects of the format and meaning of the data. A well-known example from outside the collaborative domain is the Semantic Web [Berners-Lee et al. 2001] which is defines a broad range of representations for information useful to Web services. Of course, new needs require changing the ontology which necessitates updating the applications themselves.

Many systems from the ubiquitous computing community have used context modeling techniques that are ontologically oriented. Many of these ontologies are expressed through the same building blocks as those used by the Semantic Web community, including the OWL Web ontology language [Hori et al. 2003]. Such approaches are natural, given the need for cross-application sharing and synergy. For example, systems such as CONON [Wang et al. 2004] define a fixed hierarchy for location (OutdoorSpaces and IndoorSpaces are specific types of locations; IndoorSpaces further decompose into Buildings, Rooms, Corridors, and Entries). As these authors note, however, due to the evolving nature of context-aware computing, completely formalizing all context information is likely to be an insurmountable task. These problems are inherent in using a predefined structure to try to capture a fluid and evolving notion such as context. And while some ontologically-based systems allow the embedding of application-specific data—essentially, a back door to support data outside the scope of the ontology—any such application-specific data is once again generally usable only by instances of that one application. The additional semantics encoded into it are unavailable to other tools.

Intermezzo represents a design exploration between these two alternatives of emergent structure (which limits cross-application sharing and synergy) and completely preordained structure (which limits evolvability). A key thesis of Intermezzo is that clients' agreement on a few basic structural properties of the data, coupled with a sufficiently rich data model, provide a degree of synergy and cross-application leverage missing in more loosely-structured data models, while supporting the ability to evolve to unforeseen application needs that is missing from ontological approaches. This article explores how these facilities can be used to support novel forms of contextually-enhanced coordination in collaborative applications.

At a high level, Intermezzo is an infrastructure that provides both a data store and an integrated notification service. Thus like PlaceHolder, it follows the gatekeeper pattern defined by Ramduny et al. [1998] However, unlike

PlaceHolder and Elvin, and also unlike other non-notification service systems such as Placeless Documents, the Event Heap, Linda, and so forth, Intermezzo provides a data model that is more structured than these free-form models.

As noted earlier, however, simply imposing structure is not sufficient. Structure on its own may enable machine processing and use of information, but potentially at the expense of adaptability, appropriability, and evolvability that are the forte of loosely-structured systems. In short, we need to balance the need for structure with the need for expressiveness in the data model. Thus, Intermezzo couples the basic data structures it imposes with a number of facilities described later in this article for allowing rich representations of data that can be extended by applications in ways that do not break compatibility. Structure provides the basis for cross-application agreement, while expressiveness provides the basis for adaptability that is often missing from rigorously structured approaches. By using this approach, Intermezzo attempts to support applications' interpretations and allow these interpretations to be shared rather than putting the interpretation up front in the definition of an ontology.

3. CONTEXT IN INTERMEZZO

At the core of Intermezzo is a shared object model with shared data objects called *resources* that store information. For the most part, this object model is similar to a simple classless distributed object system. Any resource can have any number of named slots that can contain simple data types or references to other resources. Resources are published into a shared dataspace by applications. Each Intermezzo dataspace represents an administrative boundary much as a place in the PlaceHolder system. The resources within a dataspace are visible (modulo access controls) to the set of clients that can access the server hosting that dataspace.

All Intermezzo resources maintain a notion of their ownership as well as access control rights. Intermezzo resources can also be replicated with variable consistency guarantees. While not discussed here in great detail (see [Edwards 1995] for more information), these facilities allow resources to be associated with users and to be securely copied and shared.

The system also supports the ability of applications to extend the runtime infrastructure through downloadable code, expressed in the Python [Van Rossom 1995] language. This feature is used by applications to impose certain semantic constraints on the data store as will be described later.

Intermezzo supports a number of operations to retrieve information from a dataspace. The most basic operation is the ability to issue queries to select a subset of the resources in the space based on some search criteria (including boolean combinations of comparison primitives against slots such as equals, less than, and refers to). The ability to search based on the contents of a resource is essentially identical to the content-based matching facilities in Elvin as well as in many tuplespace systems. These operations are designed to allow applications to be able to easily select, for example, all resources with slots that match a certain value, all resources belonging to a given user, all resources that have slots that refer to a certain resource, and so on.

Similar to the polling-based query operations, applications can submit solicitations requesting that they be asynchronously notified when some change occurs in the dataspace. These solicitations use the same content-based specification as do queries and allow applications to be notified of additions or removals of resources matching certain patterns in the data space or changes in the slots of particular resources.

So far, the features presented here are similar to those in a simple object database or tuplespace system and also similar to the data models provided by notification services. Intermezzo goes beyond this basic data model in two ways, however. First, it imposes a top-level structure on resources to facilitate the organization of contextual information. Second, it adds a set of features to allow richer representations of the data within that structure. The next two sections describe these features.

3.1 Structuring Context Through Activity

Information about users' context is stored as resources in an Intermezzo shared dataspace. Rather than allowing applications to store context information in a completely free-form ad hoc way, however, the infrastructure imposes a coarse-grained organizational structure on the context it stores.

This organizational structure is centered around the *activities* of the users of the system. Activities are represented as resources each containing references to three other resources representing, respectively, the user in the activity (called the subject), any task or application he or she is using (called the verb), and the data being operated on by this activity (called the object) [Edwards 1997]. Each of these resources may have slots containing information relevant to that aspect of the activity. For example, subjects may have slots indicating the user's identity, location, contact information, and so on. Figure 1 shows an activity resource with references to subject, verb, and object resources, each of which may, in turn, have references to other resources or simple data types such as strings.

Any given user may be involved in any number of concurrent activities, and it is the responsibility of all Intermezzo-aware applications to maintain the representations of their users' activities. Thus, applications update resources in the shared dataspace to represent changes in a user's situation—a new activity starting, a change in the user's location, and so forth. Each application is responsible for maintaining the small piece of the global activity space that it represents or knows about.

In this model, each application may have the domain knowledge appropriate for creating a particular piece of the overall picture of a given user's set of activity resources—a file editing application may know that the user is working with a set of files, while a voice chat application may know that the user is currently speaking to another party. Applications with particular domain knowledge can contribute to portions of the larger picture, but no single application need understand the entirety of the contextual information. Applications with similar semantics (file editors, e.g.) may overlap in the information that they use, update, and understand (information about files, perhaps). Other

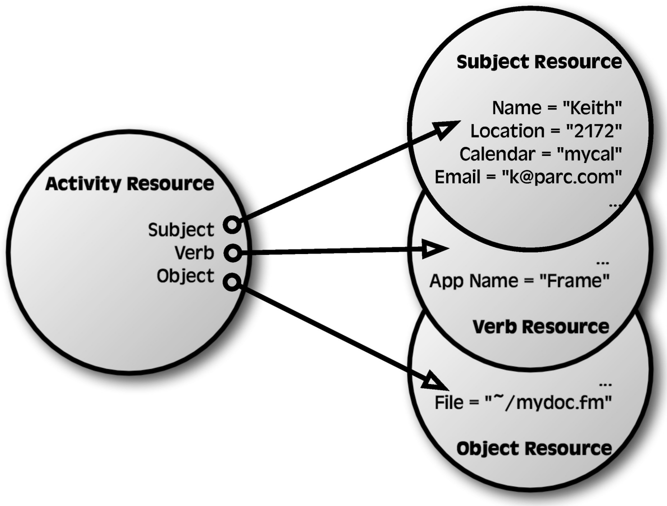

Fig. 1. A simple activity resource and its components. Arrows indicate slots containing references to resources. Other slots contain simple values.

applications may coexist with these, updating other portions of the activity space without having to understand any particular semantics of files or editing.

Of course, not all context is situated in a particular application. Some aspects of context may span a number of applications or may be external to any application at all. For example, the physical location of a user is often a very salient aspect of context, but it is outside the purview of any traditional application. So, in the Intermezzo model, a class of applications called monitors are used to monitor both the physical and the virtual worlds. They can also publish information or update existing information on behalf of applications not written to the Intermezzo APIs. This technique can be used to integrate sensing infrastructures such as those provided by Active Badges [Want et al. 1992] or the Context Toolkit [Salber et al. 1999] into the Intermezzo data model.

While activity is not the only way in which a data model for context might be organized, this simple model affords applications with enough structure to be able to share information about certain aspects of users' situations that are sufficient for a number of interesting collaborative features. The next sections of the article outline a number of mechanisms for how this simple structure can be extended by applications to accommodate their evolving needs.

3.2 Richer Representations of Context

So far, the data model as described is fairly straightforward. At the most basic level, it resembles an object database with notifications. On top of this is layered a simple structure (activity resources and their components) designed to provide a basic layout of the data in the system. This simple structure in itself is not sufficient for the flexibility and synergy needed for extensible coordination, however. There are a number of additional requirements that must be met to support these abilities, while allowing the sorts of cross-application leverage and reuse discussed earlier. This section describes Intermezzo's support for richer notions of contextual *ambiguity*, *identity*, *evolution*, and *equality*.

First, contextual information is often inherently ambiguous and varied in its intended meaning. For example, a person's location may be ambiguous because of the discrimination ability of a given sensor or potential error in sensor readings. Likewise, even the intended meaning of location may be ambiguous. Do I expect the location slot on a user's resource to indicate a building room number? Latitude and longitude coordinates? To accommodate such richness, the infrastructure must be able to support multiple notions of location simultaneously, perhaps provided by different sensors, with different formats and semantics, while allowing applications to select those that they have the programming to understand.

Second, since context often represents real world entities such as people, places, and things, we must support complex notions of identity. For example, a free-form data store such as PlaceHolder would have no intrinsic notion of what a person is. Applications could, of course, use some collection of key-value pairs to represent details about a particular person. But because these semantics are situated in individual applications rather than in the infrastructure, multiple applications may each create their own private representations of a particular person not easily available to or reusable by others. Without the ability to control how representations are created and referenced (one representation for one person, one representation for one machine, etc.) across applications we limit their ability to work together without complex upfront agreements on how every resource will be represented.

Third, we must be able to easily support evolution without breaking compatibility. For example, new location-sensing technologies will undoubtedly appear in the future with their own semantics and data formats. We need to ensure that existing applications perhaps written to use the results of some earlier location-sensing system, can still function even as new, higher-resolution information is added to the data store. In practice, this means that the data model must be extensible (meaning that no fixed schemas are in place) and that applications must be able to deal with the presence of information that may be unexpected. In terms of Intermezzo, we must be able to add new data items to resources freely; applications can then use the data items that they are written to understand, while ignoring others presumably added by other (perhaps newer) applications.

Fourth and finally, we must support more complex notions of equality of data than are typically found in most collaborative data stores and notification services. Contextual information will often represent real world entities, and such entities have complicated notions of equality. The data store must allow applications to impose semantics on the data store that reflect these notions of equality rather than relying on some simple matching of key-value pairs.

For example, if I am in an office and I ask if a coworker is here, do I mean whether the coworker is the in same office? The same building? The same city? The correct answer depends entirely on the application and the situation. For a single display groupware application, the fact that my coworker is in the same room is probably the correct interpretation, whereas for an automated away board, whether the person is still in the building is salient. The point is that equality is a complex notion, and thus the data model must support the ability of applications to create such complex, multilayered notions of equality without requiring that the infrastructure itself understand them (e.g., without understanding that offices are in buildings which are in turn in cities).

Intermezzo provides a number of special features intended to support these requirements, and enhance coordination through context.

—Evolution. Extensible slot model

—Identity. Canonicalization of resources based on real world referents

—Ambiguity. Multivalued slots

—Equality. Scoped slots

The sections that follow discuss these features of the data model and how they support representations of context. Then paper presents two examples of building on these data model features to provide direct coordination support to applications.

3.2.1 *Support for Evolution: Extensible Slot Model.* As mentioned earlier, Intermezzo-aware applications fill in the slots of published resources, representing users and their activities, with information that may be useful to them, to other applications, or to users. For example, applications may update the subject resource with contact information such as email address and telephone number or information about the user's location.

Future applications, however, may need to associate different sorts of data with users or other entities in the shared dataspace. These applications may need to store data in slots that other applications do not recognize or understand. For this reason, the Intermezzo object model is schemaless, that is, there is no type system such as class-based typing or database schemas that controls the slots on a resource. This model has similarities with prototype-based object systems such as Self [Ungar and Smith 1987] which also do away with the notion of classes in favor of schemaless instances. In such models, instances comprise named collections of slots; new instances can be created by cloning existing instances, and then modifying, adding, or even removing slots. Data elements (those things referred to by slots) are strongly typed at runtime but have no compile-time type checking.

As an example, a new application that stores Instant Messaging handles with users can just store this information in a new slot on the subject resource. Existing applications can detect the presence of the new information, but the presence of it does not change the type of the resource since it can still provide values for the slots the existing applications were written to use.

Since resources need not adhere to a strict schema, applications can flexibly add slots to resources for their own purposes and without interfering with data placed there by other applications. This allows multiple applications to associate information with subjects without colliding with each other or having to force-fit their information into a fixed schema decided upon a priori by the infrastructure designer. Obviously, while applications may not be able to interpret new slots they weren't specifically written to understand, they can still interpret the slots they were written to use. As long as the slots expected by an application are present, additional slots can be added without breaking compatibility with that application. The benefits of such arrangements have also been shown in tuplespaces as well as in systems such as Placeless Documents in which multiple applications can use the same data and coexist, sharing commonly understood data elements while ignoring others.

3.2.2 *Support for Rich Forms of Identity: Canonicalization of Referents.* A problem that arises in free-form data stores is the lack of semantics that govern object identity. For example, data models such as PlaceHolder have no intrinsic notion of what a person is and thus provide no special support for ensuring that there is one (and only one) representation of any given person in the system; applications are free to create key-value pairs representing such information on a virtually ad hoc basis. Because these infrastructures do not make any special concessions to the semantics of the data they store, multiple applications may each create their own private representation of a particular person (or other entity) that would not be available to other applications.

As mentioned before, in Intermezzo every activity in the system is represented by a separate activity resource with a reference to the subject involved in that activity. If every application were to create its own representation of the subject, then multiple subject resources would exist in the dataspace for any given person. This produces a situation in which there may be multiple, separate representations of the same real world person that may or may not be shared among applications.

An ontological solution to this problem would be to preordain a unique identity key for users and require that all applications test whether a given user's subject resource exists before creating a new one. This approach, however, has the usual limitations of ontological approaches, that is, it requires that the knowledge of what constitutes sameness be hardwired into all applications up front, that it not change, and that all applications follow the rules explicitly. Also, even if we can agree on the conditions of sameness for data objects representing users, this does nothing for other sorts of data objects with complex identity semantics that may come along in the future.

Intermezzo's approach is to allow applications to impose custom semantic constraints on the creation of resources to support complex identity semantics. Applications can extend the Intermezzo data store to allow it to canonicalize references to resources that represent (or proxy for) external entities such as users or files or processes. Support for a handful of such external entities are built into the system, and applications with novel semantics can extend the runtime system to add new canonicalization rules for resources that act as proxies for such entities such as locations, files, physical objects, and so on. The core infrastructure is able to support rich identity semantics through this feature without requiring that it be hardwired to know what users or locations or files are.

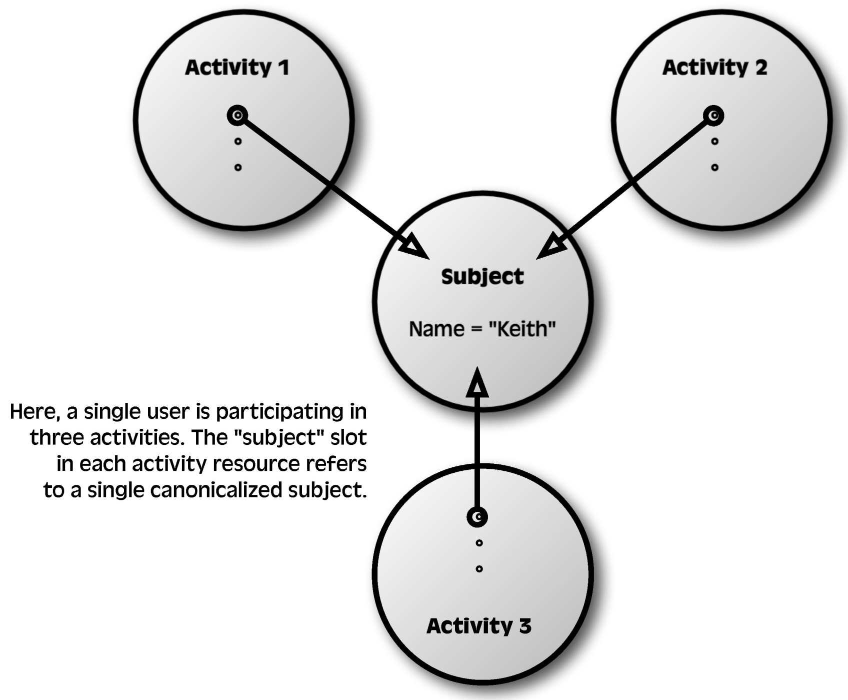
Activity 1
Activity 2
Subject
Name = "Keith"
Here, a single user is participating in three activities. The "subject" slot in each activity resource refers to a single canonicalized subject.
Activity 3

Fig. 2. Three activity resources share a canonical subject.

The canonicalization feature allows notions of identity in the virtual world to be closely aligned with notions of identity in the real world. For example, it is used to dictate that every human user of the system should have one subject resource that is the canonical representation of that person. Of course, this one subject resource may be referenced by any number of activity resources if that person is involved in multiple simultaneous activities. But the system must ensure that if these activities refer to the same individual, then they refer to the same subject resource. Likewise, the same notions of equality need to be preserved for other resources that are meant to act as representations for external entities whether in the physical or virtual worlds. Figure 2 shows three activity resources sharing a canonical representation of a user.

Canonicalization is supported through downloadable rules, expressed in Python, that govern the creation of new resources. Each rule specifies a predicate that encodes the semantics for identity, along with a list of slots that will be used in the determination process. These rules essentially allows applications to change the infrastructure's notion of what constitutes sameness for a particular type of object. Canonicalization rules are stored by the infrastructure where they are organized based on the slots they examine to determine identity. When an application attempts to create a new resource, the infrastructure first creates a temporary version of it, along with any initial values for its slots. This temporary resource is then passed to the relevant canonicalization rules which may examine its slots to determine if a slot that ascribes identity is in use and, if so, whether a canonical version of that resource already exists. If a predicate is satisfied—meaning that an application is attempting to create a new instance of a resource for which a canonical version already exists—then the predicate returns the canonical version, and the temporary resource is discarded. If no identity predicate is satisfied, then the temporary copy is allowed to stand. Applications can extend the canonicalization rules beyond the handful of ones built into the system, although this is rare. Because canonicalization rules are determined by application semantics, they can be extended over time as new constraints and conditions arise. Further, all applications that access the shared dataspace are able to reuse the effects of canonicalization without having to understand the specific rules for a particular type of object.

Note that canonicalization is not simply the ability of slots to hold references to resources. Even with references, we would need to have a systematic way to control the creation and use of these proxy resources. Instead, canonicalization controls resource creation and reference to ensure behavior according to an extensible set of rules established by applications to govern the mapping of virtual resources to physical resources. Perhaps more importantly, once the canonicalization rules for a given resource are in place, it is transparent to applications; they acquire the appropriate canonicalization behavior for free.

3.2.3 *Support for Contextual Ambiguity: Multivalued Data.* Intermezzo allows a single slot to contain multivalued data. This means that slots can refer to a collection of zero or more data elements or resource references. The collection of objects referred to by a multivariate slot is unordered and possibly empty.

This feature is used as a way to explicitly represent ambiguity in the system. For example, different location-sensing technologies may have different accuracies, latencies, and so on. The notion of a user's location may not be a single, reliable value but rather a collection of values from different sources, and perhaps with different confidences associated with each source. Ambiguous interpretations are supported explicitly without requiring the infrastructure to decide on a single correct one, and hence having to know about the meaning of the information, the characteristics of various sensing technologies, and so on. Such multiple interpretations can then be presented to applications or users.

The semantics of the query operators is extended for multivalued data. There is a weak equals operator which essentially means contains, and a strong equals operator which means that every item of the two operands must match. (For univalued data, weak and strong equals are the same operation.) Intermezzo provides two assignment operations to support both the single and multivalued slots. A replacement assignment operation overwrites a single-valued slot with a new value and causes an error when applied to a multivalued slot; an additive assignment operator adds a new value converting a single-valued slot to a multivalued one if necessary.

While a multivalued data model may be useful for representing ambiguity, the inherent lack of structure provided by such models make them problematic for representing data that may not be ambiguous but is simply multifaceted by nature. Intermezzo's scoped data model, described in the following, can be a more effective way of representing such data.

3.2.4 *Support for Layered Interpretations of Equality: Scoped Data.* Finally, Intermezzo introduces a unique type of data model used to represent information that can naturally be viewed at multiple granularities or scopes. The metaphor here is one of magnification—certain applications may care about fine-grained details of the information, while others may care about more global details. Applications can choose the granularity at which they view and interpret contextual data, and multiple granularities of interpretation can coexist simultaneously.

To motivate scoped data, consider how one would represent the object of a certain action such as editing a file. Clearly the file itself is the object, in a sense. But many more aspects of this file may be salient in different situations. A system management tool may care about the fileserver on which the file resides, a source code control system may care about the directory in which the file resides, and a collaborative editing tool may care about the user's individual position within the file.

All of these aspects of the object may be salient in different situations, and there may be other potentially useful aspects that are unknown at the time these applications were written. Because of this, we cannot simply make an a priori decision to hard code certain aspects of files in an attempt to capture all salient features of them. Such a strategy would certainly not be flexible enough to adapt to new sorts of applications that come along in the future. And such a strategy would also require us to do the same sort of semantic analysis of the features of other types of objects that might potentially be useful in the future. Instead, what we need is a means of representing multiple aspects of an object, while allowing the dynamic addition of aspects representing new interpretations.

This is the intended use of Intermezzo scoped data: to provide dynamic multi-granularity interpretations of data. Essentially, it is a variation on simple multivalued data where each value of the collection has a named scope associated with it that denotes its granularity in the collection. For example, in the file example, the scoped data might contain an identifier for the file in the file scope, an identifier for the directory (or directories) containing the file in the directory scope, the name of the host containing the file in the fileserver scope, and so on. Each scope represents a particular aspect of the object.

This model captures the notion that, when working with a file, an editor application is not just working with that file alone. It is also implicitly making use of the directory containing that file, the fileserver that the file resides upon, and so on. Even though these other interpretation of the object of the editing application may not be important to it, they may be important to other tools operating in that contextual fabric.

There are two important aspects of this arrangement. First, the set of scopes for a data element is not fixed ahead of time and so new scopes, or interpretations of existing data, can be added after the fact. And second, the scoping mechanisms allow domain-dependent and domain-independent interpretations to coexist in the same data object.

So, for example, a user may be using a project management tool that publishes activity data referring to a given file, perhaps adding interpretations

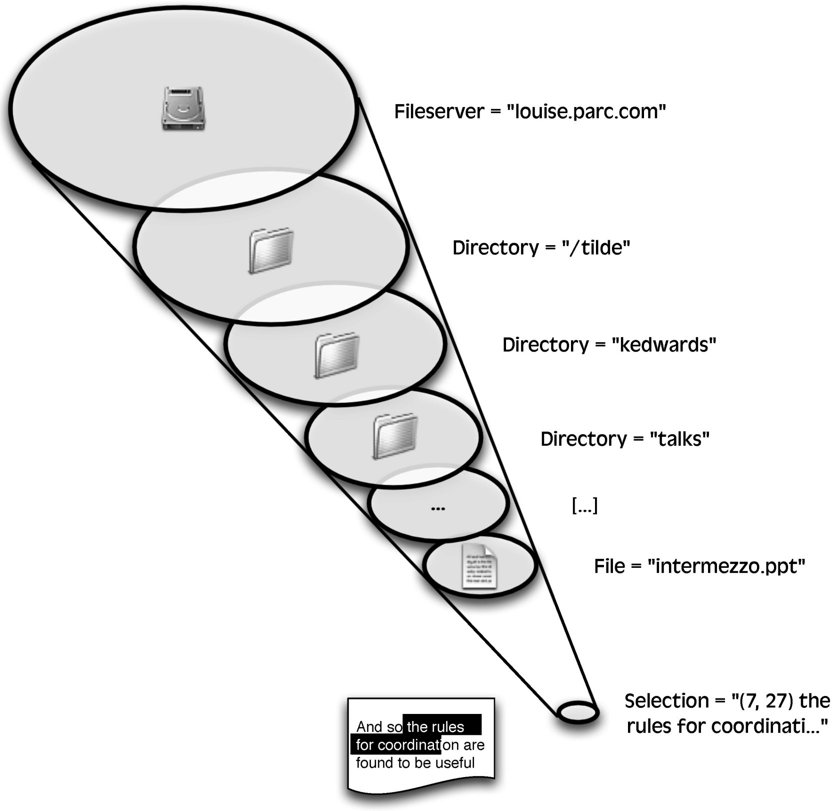
Fileserver = "louise.parc.com"
Directory = "/tilde"
Directory = "kedwards"
Directory = "talks"
...
[...]
File = "intermezzo.ppt"
Selection = "(7, 27) the rules for coordinati..."
And so the rules for coordination are found to be useful

Fig. 3. A scoping stack for a file object.

within the file, directory, and fileserver scopes. These are all domain-independent interpretations that any application that understands files might be able to add or use. If the user is concurrently using a collaborative editing tool to work with that same file, this tool would publish a new activity record referring to the same file object (using the canonicalization mechanisms already discussed). This tool, since it understands the internal structure of the file, would then add a new scope to that object—in effect, saying that not only is it working with this given file, but that the user is currently working in a particular portion of that file. In this case, the information added by the editor is domain-dependent, that is, it would only be meaningful to similar applications equipped with the domain knowledge to interpret information about the selection. The approach is similar to that of Patel and Kalter [1993] but is generalized to support new data types.

Figure 3 shows what such a scope stack looks like for a file. Here you see a number of domain-independent interpretations of the object (fileserver, directories, the file itself), as well as a domain-dependent interpretation of the selected region inside the file.

This is an example of a set of applications colluding to produce a richer picture of the object in use than simply file. These applications share the same data objects but interact with those portions that are meaningful to them and for which they are equipped with the semantics to use. In this example, both the project management tool and the collaborative editing tool would know that users happen to be interacting with the same file since both are semantically equipped to understand and look for the file scope. Information particular to the collaborative editor's notion of selection would be ignored by the project management tool since it doesn't understand (or presumably care about) such aspects of the files it manages.

The underlying data model supports these multiple, layered interpretations of the data without having to embody any semantic knowledge of what a file is, or the fact that it lives on a filesystem or in directories, or that some of them might contain elements that can be individually manipulated. Domain-independent layers of interpretation can coexist with domain-dependent layers added by applications with specialized domain knowledge (like the editing tool mentioned earlier). All of this is done without requiring any understanding on the part of the data model and without all applications having to agree in advance on a set of common aspects of files (or other objects) that they might find useful.

Scoping could be achieved by having applications use a coordinated name mangling approach to naming slots (for example, by having applications use slots named "file_directory," "file_host", "file_selection") rather than using the file scoping mechanism described. Such an approach relies on applications to agree on the name mangling procedure and implement it and comparisons of scoped data correctly in all cases. Intermezzo instead embeds this functionality directly into the runtime, making it a first-class data type and supporting it with a number of operations designed to take multigranular views of data into account. An intersection operation, for example, can be used to indicate what aspects two objects have in common. An intersection of object resources representing two different files might indicate that, for example, the files reside on the same filesystem and in the same directories.

Simple equality comparison simply determines whether the aspects in one object are a proper subset of the aspects in another. The semantics here are meant to indicate, for all the aspects of this object that I care about, is it the same as this other object. Two objects are equal if and only if, for all of the scopes in the first object, the values for those scopes are the same in the second object.

A different selective equality operator requires three operands, namely, the two scoped objects to compare, and a primary scope which is to be considered the most salient for purposes of comparison. The system extracts the values of the primary scope from both objects and then compares them to determine selective equality. While all of these operations could be implemented in each application, placing them in the runtime allows a consistent view of scoped data across applications.

3.3 Summary

The activity-oriented structure imposed by Intermezzo provides applications with the basic means for talking about actors and activities in the system. It allows them to ask questions about what tasks a set of users are engaged in, what artifacts they are working with, whether users are working with the same artifacts, and so on. Beyond this basic structure, however, a number of features in the data model support its adaptation and evolution by applications. Applications can embed new semantics into the data model that change the way the answers to questions like the ones just described are answered. As the set of applications in use changes, new information is added to the system, and new interpretations appear.

In other words, the structural convention of activity resources provide the shared agreements that guarantee that applications share some common language about context. The four features outlined in the previous sections—extensible slots, canonicalization, multivalued data, and scoped data—facilitate the expression and sharing of complex application-driven semantic structure to represent context. Together, these aspects of the infrastructure distinguish it from a more conventional data storage system and also distinguish it from ontology-based approaches that require that all such conventions and structures be embedded in the applications ahead of time.

The primary benefit of this middle-of-the-road approach is that applications written only to understand the coarse-grained structure can benefit from the work of other applications in expanding the semantics of the data stored in the infrastructure. For example, applications can be written very easily to detect whether two users are engaged in activities that involve the same objects (files, or machines, e.g.) simply by testing whether the object resources are equal and without having to understand the intricacies of all of these various data types nor what equality means for them. Other applications which do have these understandings can embed the necessary semantics into the infrastructure, allowing it to be leveraged by other applications. (The next section shows examples of how this works and how it can be useful for coordination.) The layered interpretations in the infrastructure can thus grow over time as new applications appear that are equipped with the programming necessary to augment the infrastructure for all other applications.

While this combination of coarse-grained structure and rich representation does not, of course, solve all problems with agreement, it does allow the creation of applications that need only understand the basic structure and yet reap benefits from work done by other applications.

The extensible slot model allows new aspects of data to be hung on the basic activity structure; applications can augment the slots that comprise any of the resources in the system over time without requiring change in the infrastructure. Much like tuplespace systems, this extensibility allows applications to share parts of the contextual state that they are equipped to use while ignoring other unknown parts.

Ambiguity is exposed in the data model through the use of multivalued data types. Such representations do not force a preferred interpretation at the expense of other possible interpretations. Multiple interpretations of a data element, perhaps placed there by different applications, can coexist simultaneously.

Resource canonicalization is essential in a system that's meant to provide a virtual mirror onto aspects of the physical world. In the physical world, we have strong notions of identity which can easily be lost in the virtual world. Canonicalization ensures that virtual objects that are meant to have the semantics of being proxies for real world objects maintain consistent notions of identity. The infrastructure itself need have no particular knowledge of the physical meaning of subjects or files or locations as applications can extend the canonicalization behavior of the system by adding new canonicalization rules for each type of resource.

And finally, scoped data are a way for applications to provide multiple layers of interpretation of a given object. Different aspects of a given object may be salient to different applications, and we cannot determine in advance what aspects are important for all types of objects. Scoped data provides a way for multiple interpretations to coexist; applications share the interpretations that they're equipped to understand and ignore those that they are not. The infrastructure, again, need not understand any of these interpretations or even for that matter know that they exist since applications are free to dynamically add new interpretations as needed.

The next section of this article explores how these features are used in practice through two examples of context use in Intermezzo: a context-enhanced session management facility and a context-enhanced access control system. These services represent higher-level building blocks that are layered on the lower-level contextual data representation facilities and use the contextual information available there. These services can by used by collaborative applications to make them more responsive to the situations of their users.

4. APPLYING THE INTERMEZZO DATA MODEL

This section shows how the flexible data model used by Intermezzo, coupled with the structure afforded by the activity representations, can be used to build higher-level coordination features that can provide context-enhanced services across applications. This section describes two such services, context-enhanced session management and context-enhanced access control. These are services that can be used to make collaborative applications responsive to context in particular ways. As these have both been previously presented in the literature, this section provides only brief descriptions of how these features have been constructed using the basic facilities offered by Intermezzo and how they can assist in cross-application coordination.

4.1 Context-Enhanced Session Management

This section describes one of the higher-level building blocks that has been implemented on top of Intermezzo: a context-enhanced session management service for initiating interactions among users (this work was first described by the author in Edwards [1994]). Unlike most traditional session management services, the one provided by Intermezzo has been extended to take advantage of context to provide more lightweight and fluid forms of interaction than have traditionally been found in collaborative systems. Applications can use this service to enhance their own interactions with users.

Most collaborative applications take one of two heavyweight approaches to session management. In *initiator-based* systems, one user invites others to an interaction through a series of dialogs (see, for example, MMConf [Crowley et al. 1990] and RTCAL [Greif and Sarin 1987]). In *joiner-based* systems, an initiating user creates a new session and users find the session either by browsing, or knowing a priori that the session is taking place, along with some name or handle for it. Examples of systems that follow this model include IRC [Oikarinen and Reed 1993] and instant messaging [Grinter and Palen 2002]. Participants typically use some other out-of-band means to notify each other that a collaboration is in progress (often a telephone call).

In contrast to these *explicit* models of session management that require action orthogonal to the task at hand to join a session, context enhancement provides the opportunity for a more lightweight, ad hoc approach to session management. This lightweight approach might be called *implicit* session management because the acts of rendezvous and interconnection are inherent in the very act of interaction. For example, when two people are working with the same object or artifact, there may be a potential for collaboration. By virtue of the actions in which they are engaging, we know that both, at least for the moment, share an interest in some aspect of the artifact they are interacting with. In implicit session management, the opportunity for interaction exists purely because of the work that you happen to be doing anyway not because of any explicit external actions taken.

The facilities provided by Intermezzo can provide a basis for a rich and extensible context-enhanced session management service. Further, these facilities allow the creation of such services that can detect confluences of activity and other aspects of user context without requiring that these services understand the semantics of such context.

A separate session management service built using Intermezzo monitors the contextual model to look for overlaps or confluences in the context data published by the applications on the network. When two activity tuples exist that contain objects that are equal, then Intermezzo's session management service can notify the applications represented in those activity tuples (or another, third application) to allow the users to enter into a collaboration.

For example, if two users edit the same file, the session management service can notify the involved applications of this fact which may then allow them to enter into a spur of the moment collaboration. The mechanics of joining a collaborative endeavor thus closely match the human dynamics of collaboration where, if two coworkers wish to work on a budget together, they simply meet up to work on the budget at roughly the same time; no formal invitations are issued, and no name must be given to the activity. Unlike explicit forms of session management, where the burden of rendezvous is on the users, implicit context-enhanced session management allows the system to assume the burden of detecting and notifying potential collaborators.

Both the structural conventions and the rich representational facilities in Intermezzo are necessary to fully support this style of session management in a flexible and extensible way. Applications that wish to be informed of confluences in activity simply register to receive notifications when two activities exist that refer to different subject resources but the same object resources. The ability to express this event subscription relies only on the structural conventions of activity resources. And yet the representational features leverage the varied interpretations layered into the data model by applications, and allow the notion of sameness of objects to mirror the semantics of real world objects and to evolve in rich ways. As new types of artifacts are modeled by applications in the system (new scoping rules are provided by applications, new details are added via new slots, etc.), the session management service can take advantage of these richer semantics without having to have explicit knowledge or understanding of what any particular types of objects mean.

As an example of how the structural and representational features together support cross-application coordination, consider a shared document editor and a source code awareness tool. The shared document editor may only care about users editing the same file. But the source code awareness tool, which notifies users based on edits on all files in a folder, can still coordinate with the shared document editor—the scoped representation for file objects ensures that the two applications can be notified about confluences in user context even though one is interested in files, while the other is interested in the directories in which those files exist.

While the currently implemented session management service only determines potential interactions based on the sharing of artifacts, other sorts of potential interaction triggers could exist as well. For example, other interaction criteria such as colocation can also benefit from these multivalued, extensible representations. Applications that care about colocation of users can simply solicit events when two activities exist for which the user's (potentially multivalued) location attributes are the same. Additional sensors can add new interpretations to the location, perhaps representing longitude and latitude coordinates from a GPS system, room number from an indoor tracking system, and so forth. Applications can be extensible to such new location sensors and new data formats for storing the information by simply soliciting to be notified about confluences in location and without needing to know the details of the various sensors.

The role of the infrastructure in this approach to session management is merely to detect confluences in contextual information and then notify applications. The infrastructure itself does not make any judgement as to the appropriateness of any particular confluence in signaling the desire for interaction. It does not, for example, automatically decide that the users wish to collaborate just because they happen to be sharing some artifact. Instead, that decision is left to applications to make based on their semantics. Even though an application receives a message from Intermezzo indicating that a potential collaboration exists, the application is not required to act on it; it may prompt the user or ignore the message entirely.

4.2 Context-Enhanced Access Control

Clearly, there are issues involved in providing mechanisms for capturing and disseminating potentially sensitive information such as a user's context.

Security systems protect information by using schemes such as access control lists and capabilities to determine who is allowed access to certain information. But these systems are often heavyweight from the user's perspective especially in the context of collaborative applications. For example, as Neuwirth et al. [1990] note regarding the domain of collaborative writing,

> There is a potential problem in systems which support the definition of social roles: "premature" definitions of these roles can lead to undesirable consequences. For example, it is not always clear at the outset of a project who is going to make a "significant contribution" and therefore who should get [the] authorship [role]. But if authorship is defined at the outset, then it may reduce the motivation of someone who has been defined as a "non-author" and the person may not contribute as much.

Others, such as Dewan et al. [1994] and Dourish and Bellotti [1992], have registered similar concerns that role-based access control systems, because they tend to be relatively static and often involve explicit heavyweight operations to switch among roles, may limit fluidity in collaborative settings.

Again, this is a situation in which a more contextually sensitive form of coordination may be useful. Thus, the features in Intermezzo have been applied to create a *context-enhanced access control* service (described more fully in an earlier paper [Edwards 1996]). This service uses the data store as well as the underlying access control facilities described in Section 3 to allow access control decisions to be based in the context of the users of the system and their activities.

This access control system adapts certain terminology and features from traditional Role-Based Access Control (RBAC) [Sandhu 1998] models in which *roles* represent sets of users and *policies* specify the set of rights for accessing resources. In many traditional collaborative systems, the binding of users to roles is fairly static. The problems related to such premature role membership have been identified in the literature [Beck and Bellotti 1993; Neuwirth et al. 1990] and include the fact that membership in a given role is typically predefined and fixed for certain access rights. Users must anticipate role membership and take responsibility for updating it as appropriate, and roles can only be defined in terms of user names (or other representations of identity) not other attributes of users or the environment.

Unlike these static systems, however, the context-enhanced access control service allows the determination of membership in a role to be based on the context of a user, or any other context stored in the system, rather than in some predetermined membership list. Further, the system can evaluate a user's access rights dynamically (as requests for access are made) to allow applications to create access control policies that regulate the use of context and that are themselves responsive to the contextual states of users.

This runtime assignment of rights is managed through the notion of *dynamic roles* which indicate a set of users not though an explicit membership list but rather through a description of the contextual attributes of those users. Applications can create new dynamic roles that reflect their particular semantics and can express the criteria for membership in these roles through rules about how context should be used to control access to the information they publish. These rules take the form of predicate functions which can use the contextual information stored in the dataspace to make decisions about whether to grant or deny access to resources in the space. Access rights to slots on resources are granted to users in certain roles; whenever a given user agent attempts to access a slot, Intermezzo evaluates the predicate that determines whether that user should be considered a member of the role, at runtime. The system also supports traditional statically-defined roles.

The use of dynamic roles is reminiscent of work by others in the security and database communities [Blaze et al. 1996; Wong et al. 1997; Woo and Lam 1998]. In the CSCW literature, the approach is somewhat similar to the abstract roles of Kanawati and Riveill [1995], although their abstract roles are less expressive than Intermezzo's dynamic roles. (Kanawati and Riveill's abstract roles are based in formal a priori construction of a graph of organizational access rights and seem designed primarily to modularize access control specification by segregating the rights afforded to groups. They do not allow runtime evaluation of arbitrary predicate code nor do they support the use of contextual data in evaluating access rights.)

The use of potentially arbitrary predicates to determine membership lends expressive power to dynamic roles. Predicates are expressed in the Python language and are uploaded by applications to the Intermezzo server infrastructure where they are evaluated and execute with the permissions of the user requesting access. The predicate is free to query the state of the dataspace to determine whether to grant access. Predicates can examine conditions such as time of day, location, physical colocation, shared resources, and so on, using the same operators as available to normal application code. (They therefore can take advantage of expanded notions of equality for scoped data, canonicalized resources, etc.) In order for access to be granted to a given resource, all of the access predicate functions that apply to that resource must be satisfied (in other words, applications cannot grant themselves access to a resource simply by using a role that would give themselves access to a given resource if such access would not be granted otherwise). Access rights can be assigned to resources at the time of their creation.

By moving the determination of membership from session startup time to evaluation time and by using predicates rather than membership lists, dynamic roles acquire several interesting properties.

—Dynamic roles allow membership to be based on any contextual attribute of a user not just the user's identity.
—Potential membership can vary from moment to moment during the lifetime of a session.
—Access can be granted based on the instantaneous state of the user's world.
—By describing role membership rather than specifying it, users can be relieved of some of the burden of tracking, updating, and anticipating role membership explicitly.

As an example, dynamic roles allow the specification not only of "people who work in my lab," but "people who are in my lab right now" as a category of users. These rules are provided by applications which presumably are equipped with the domain knowledge necessary to establish informed guidelines about access control.

Similar forms of access control based in contextual attributes have been explored by others such as McDaniel's Antigone Condition Framework [2003]. McDaniel's work acknowledges the importance of the ability to use arbitrary programming code to determine whether an access control policy is satisfied and implements dynamically loadable code to allow the creation of predicates analogous to the use of application-supplied Python code in Intermezzo. In other ways, however, the Antigone system is complementary to the system described here—Antigone provides an alternative framework for expressing arbitrary conditional code but does not provide a data representation layer analogous to Intermezzo's.

Like session management, these structural and representational features of the Intermezzo data model support separation between semantic interpretation (provided by applications) and enforcement of access rights (provided by the infrastructure). Applications implement predicates that reflect their specific semantics with regard to access control. It is these predicates that are responsible for any understanding of particular contextual aspects that may be required by the application and provide a way for applications to express yet another aspect of their semantics into the shared dataspace. Essentially, these predicates provide a way for applications to extend the systems' low-level access control primitives with code that reflects their particular interpretations of contextual information. This interpretation is provided piecemeal by applications and the infrastructure need not be aware of it.

As noted earlier, predicate code uses the same operations over the dataspace as the session management service and other applications. Thus, it can take advantage of new interpretations and extensions to the context model added by applications (additional scopes or identity rules or new slots on resources, e.g.) Conversely, since predicates control access to the contextual dataspace itself, the mechanism allows new applications or utilities to create access policies that affect existing applications that use the dataspace.

5. INTERMEZZO FROM THE APPLICATION'S PERSPECTIVE

Access to the Intermezzo runtime is provided through toolkit libraries available in both C++ and Python. In its most basic use, the library is simply used to publish activity data from the application. This use helps to maintain the global repository of context but does not support context-enhanced applications that can actually respond to changes in the environment.

In this basic publish mode, applications simply link in the library and make (typically) a single call to publish an activity record in the shared dataspace. The library automatically fills out the slots of the activity records as much as possible, including creating scoped representations of common types such as files and reusing canonical resources to represent entities such as users. Applications are free to decorate this basic representation with additional information that may be salient to them or which they are in a unique position to know. In general, the addition of information to activity records is very easy, applications simply name the slot they wish to write into, and then provide the value. Slots automatically become multivalued as additional data is written to them.

A slightly more complicated use of maintaining the contextual world view involves applications that need to create new object types with their own semantics for scoping and perhaps new canonicalization rules. These applications will generally define a new namespace to represent the multiple views of the object. For example, a mail program may create a mail namespace with scopes for message parts, messages, folders, and servers.

Even if applications do not explicitly flesh out the data they publish, this data may be augmented by monitor applications running on the network. For example, one basic monitor detects the creation of user resources and annotates them with basic contact information. A location monitor can update user resources with information about a user's location. Other applications that make use of these user resources get this additional information for free in the sense that applications can reuse the work of others to maintain an enhanced picture of users' context.

As mentioned previouly, simply publishing activity records is a minimal behavior that applications use to help maintain the global view of context. Applications that wish to respond to changes in context can also listen for updates to contextual state or retrieve information from the dataspace based on user actions. For example, a context-enhanced mail application can select out resources corresponding to the user in the from line of a selected email message, allowing the application to display information about the current user, including contact information, location, and so on. This same program can use the context-enhanced session management framework to easily detect confluences in object resources (the mail message being viewed) to show others looking at the same mail message (such as an email sent to a list). Tools such as collaborative editors can use this same mechanism to receive notifications when others edit the same files or files in the same directory, or when others are working in the same portion of a file according to their interest.

Likewise, applications can use the context-enhanced access control mechanisms to modify certain access rights according to situation. For example, a collaborative editor to support the ability of users working together on a document may reveal certain information about themselves (contact information or current availability) that may not be revealed to others.

6. SUMMARY AND REFLECTIONS

Intermezzo is an infrastructure project designed to support the creation of collaborative applications that can fluidly and extensibly leverage the context of their users. A key principle in Intermezzo's approach is that it is generally impossible for an infrastructure which by its nature must be designed for generality and reuse to be in a position to make meaningful interpretations about something as rich as human context. Such interpretation, when made by machines at all, is best made not by the infrastructure, but by applications with the particular domain knowledge necessary to act on it and with the proximity to the user to be able to defer to him or her when appropriate.

Based on such a premise, the role of the infrastructure must be to support the varied needs of these applications without constraining the unforeseen uses or interpretations they may place on context. Further, in a situation in which the global contextual view is built piecemeal by the individual applications that may have responsibility for bits and pieces of it, the infrastructure should as much as possible ensure that applications can take advantage of the work done by each other, that multiple isolated, incompatible partitions do not arise in the dataspace, and so on.

The key contributions of this work are in a set of features intended to allow a range of applications each with their own semantics, domain knowledge, and salient aspects of context to adapt the infrastructure to accommodate their needs. This is done through a storage model that allows applications to embed certain restricted interpretations into the infrastructure itself. Structural conventions are essential because they provide the common ground on which applications are written. Intermezzo assumes that most applications will be coded only to perform simple queries on the basic activity structure for example, tests for sameness of objects.

Beyond this, particular aspects of the data model allow applications coded against the basic activity structure to inherit richer interpretations of that data provided by tools that can embed domain semantics into the infrastructure. Scoping, for instance, allows multiple application-provided interpretations of an object to coexist in the data model; multivalues accommodate ambiguity; an extensible canonicalization model allows virtual identity to mirror notions of identity in the physical world; and so on.

The infrastructure gives applications, which presumably have more domain knowledge than the generic infrastructure, the responsibility for maintaining, sharing, perhaps even interpreting the contextual space, while leaving the infrastructure itself free from having to understand the semantics of context or act on interpretations of such semantics. The ability of applications to participate in such features as rich, contextually-enhanced session management, while knowing nothing of the semantics of the objects or applications participating in a collaboration, demonstrates the power of extensible, expressive representations.

As noted in the motivation section, applications already shoulder a burden of imposing semantics on top of loosely-related data stores, for instance, they must agree on which keys to use for data, what the format of that data will be, when to reuse existing data, what sameness of that data means, and so on. The difference is that Intermezzo provides the ability for applications to embed these semantics constraints directly into the infrastructure (in the form of canonicalization rules, access control rules, scoped interpretations, etc). This embedding provides two important advantages.

(1) The constraints ensure that poorly written applications don't neglect the semantic constraints of the data.
(2) The constraints leverage the balance of structure and expressiveness to allow applications to reuse the benefits of semantic interpretations provided by other applications.

A number of problems that already exist in applications that make use of loosely-structured data stores are still present in Intermezzo. Perhaps most importantly, for applications to test against individual slot values on resources, they must agree on the names of those slots (that a user's location is stored in a slot called location and not position or loc, for instance). While Intermezzo's features are geared largely toward providing a richer value space at least partial agreement on the key space is also needed. We should note that such name agreements are needed in any loosely-structured data store, and Intermezzo doesn't make the problem in this regard any worse.

ACKNOWLEDGMENTS

Many thanks are due to my mentors at Georgia Tech and elsewhere who influenced this work: John Stasko, Jim Foley, Scott Hudson, Colin Potts, and Prasun Dewan. Thanks also to Rebecca Grinter for her invaluable comments on this article.

REFERENCES

Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. 1997. Cyberguide: A mobile context-aware tour guide. *Wireless Networks 3*, 5, 421–433.

Arnold, K., O'sullivan, B., Schieifler, R. W., Waldo, J., and Wollrath, A. 1999. *The Jini Specification*. Sun Microsystems Press, Addison-Wesley Publishers, Reading, MA.

Beck, E. E. and Bellotti, V. 1993. Informed opportunism as strategy: Supporting coordination in distributed collaborative writing. In *Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW '93)*. Milan, Italy (Sept.), G. De Michelis, C. Simone and K. Schmidt, Eds. Kluwer Academic Publishers, 233–248.

Berners-Lee, T., Hendler, J., and Lassila, O. 2001. The Semantic Web. *Scientific American*.

Blaze, M., Feigenbaum, J., and Lacy, J. 1996. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA (May), 164–173.

Bly, S., Harrison, S., and Irwin, S. 1993. Media spaces: Bringing people together in a video, audio, and computing environment. *Commun. ACM 36*, 1, 28–47.

Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. 1990. MMConf: An infrastructure for building shared multimedia applications. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)*. Los Angeles, CA. ACM Press, 329–242.

Day, M., Patterson, J. F., and Mitchell, D. 1997. The notification service transfer protocol (NSTP): Infrastructure for synchronous groupware. In *Proceedings of the Sixth World Wide Web Conference*.

Dewan, P., Choudhary, R., and Shen, H. 1994. An editing-based characterization of the design space of collaborative applications *J. Organiz. Comput. 4*, 3, 219–240.

Dey, A., Salber, D., Abowd, G., and Futakawa, M. 1999. The conference assistant: Combining context-awareness with wearable computing. *IEEE Symposium on Wearable Computing (ISWC'99)*.

Dourish, P. and Bellotti, V. 1992. Awareness and coordination in shared work spaces. In *Proceedings of ACM Conference on Computer-Supported Cooperative Work*, Toronto, Canada (Nov).

Dourish, P., Edwards, W. K., Lamarca, A., Lamping, J., Petersen, K., Salisbury, M., Thornton, J., and Terry, D. B. 2000. Extending document management systems with active properties. *ACM Trans. Inform. Syst.*

Edwards, W. K. 1994. Session management for collaborative applications. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. Chapel Hill, NC. (Oct).

Edwards, W. K. 1995. Coordination infrastructure in collaborative systems. Ph.D dissertation. College of Computing, Georgia Institute of Technology, Atlanta, GA.

Edwards, W. K. 1996. Policies and roles in collaborative applications. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. Boston, MA.

Edwards, W. K. 1997. Representing activity in collaborative systems. In *Proceedings of the 6th IFIP Conference on Human Computer Interaction*. Sydney, Australia (July).

Fitzpatrick, G., Parsowith, S., Segall, B., and Kaplan, S. 1998. Tickertape: Awareness in a single line. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '98)*. Los Angeles, CA (Apr.), 18–23.

Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T., and Segall, B. 1999. Augmenting the everyday world with Elvin. In *Proceedings of the European Conference on Computer-Supported Collaborative Work (ECSCW)*. Copenhagen, Denmark. Kluwer Academic Publishers, 431–451.

Gelernter, D. 1985. Generative communication in Linda. *ACM Trans. Prog. Lang. Syst. 7*, 1, 80–112.

Greif, I. and Sarin, S. 1987. Data sharing in group work. *ACM Trans. Office Inform. Syst. 5*, 2, 187–211.

Grinter, R. E. and Palen, L. 2002. Instant messaging in teenage life. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2002)*. New Orleans, LA (Nov). 16–20.

Gutwin, C., Greenberg, S., and Roseman, M. 1996. Workspace awareness support with radar views. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)*.

Gutwin, C., Roseman, M., and Greenberg, S. 1996. A usability study of awareness widgets in a shared workspace groupware system. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. ACM Press, 258–267.

Hori, M., Euzenat, J., and Patel-Schneider, P. 2003. OWL Web Ontology Language XML Presentation Syntax. W3C Note 11-June-2003. http://www.w3.org/TR/owl-xmlsyntax.

Johanson, B., Fox, A., and Winograd, T. 2002. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Comput. 1*, 2, 71–78.

Kanawati, R. and Riveill, M. 1995. Access control model for groupware applications. In *Proceedings of Human Computer Interaction*. Huddersfield University, UK (Aug.). 66–71.

McDaniel, P. 2003. On context in authorization policy. *Eighth ACM Symposium on Access Control Models and Technologies (SACMAT)*. (June), 80–80.

McGuffin, L. and Olsen, G. 1992. ShrEdit: A shared electronic workspace. Cognitive Science and Machine Intelligence Lab, University of Michigan.

Neuwirth, C., Kaufer, D. S., Chandhok, R., and Morris, J. 1990. Issues in the design of computer support for co-authoring and commenting. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)*. Los Angeles, CA. 183–195.

Oikarinen, J. and Reed, D. 1993. Internet relay chat (IRC) protocol. IETF. Internet Request for Comment RFC1459.

Patel, D. and Kalter, S. D. 1993. A unix toolkit for synchronous collaborative applications. *Comput. Syst. 2*, 6 (Spring), 105–134.

Ramduny, D., Dix, A., and Rodden, T. 1998. Exploring the design space for notification servers. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*. Seattle, WA, 227–235.

Root, R. W. 1988. Design of a multimedia vehicle for social browsing. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*.

Salber, D., Dey, A. K., and Abowd, G. D. 1999. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, PA. (May). 434–441.

Sandhu, R. S. 1998. Role-based access control. *Advances Computers 4*, 6, 237–286.

Stewart, J., Bedersen, B. B., and Druin, A. 1999. Single display groupware: A model for co-present collaboration. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. 286–293.

Ungar, D. and Smith, R. 1987. Self: The power of simplicity. In *Proceedings of the Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*. Orlando, FL. (Oct.). 227–242.

Van Rossum, G. 1995. *Python Reference Manual*. Centrum voor Wiskunde en Informatica (CWI).

Wang, X., Zhang, D., Gu, T., and Pung, H. 2004. Ontology based context modeling and reasoning using OWL. *Workshop on Context Modeling and Reasoning*, *IEEE Conference on Pervasive Computing and Communication (PerCom'04)*. Orlando, FL. (March).

Want, R., Hopper, A., Falcao, V., and Gibbons, J. 1992. The active badge location system. *ACM Trans. Inform. Syst. 10*, 1, 91–102.

Wong, R. K., Chau, H. L., and Lochovsky, F. H. 1997. A data model and semantics of objects with dynamic roles. In *Proceedings of the 13th International Conference on Data Engineering (ICDE)*. Birmingham, UK. (Apr.). IEEE Computer Society, 402–411.

Woo, T. and Lam, S. 1998. Designing a distributed authorization service. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. San Francisco, CA. (March).

Received August 2004; revised April 2005; accepted June 2005 by Prasun Dewan