# Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment

Emre Kıcıman and Armando Fox

Stanford University
{emrek,fox}@cs.stanford.edu

**Abstract.** The original vision of ubiquitous computing [14] is about enabling people to more easily accomplish tasks through the seamless interworking of the physical environment and a computing infrastructure. A major challenge to the practical realization of this vision involves the integration of commercial-off-the-shelf (COTS) hardware and software components: consider the awkwardness of such a mundane task as exporting a textual memo written on a Palm Pilot to a Microsoft Word document. It is not enough to overcome the protocol and data format mismatches that currently impede the interoperation of these entities: for the user experience to be truly seamless, we must provide a framework for the *dynamic* connection of such endpoints on demand, to support the ad-hoc interactions that are an integral part of ubiquitous computing. To this end, we offer a dynamic mediation framework called Paths. A Path consists of dynamically instantiated, automatically composable operators that bridge datatype and protocol mismatches between components wishing to communicate. Because operator composability is inferred from the type system, adding support for a new type of endpoint requires only incremental work; because the control and data flow for Paths are largely decoupled from the communicating endpoints, it is easy to connect COTS or legacy components. We describe the Paths architecture, our prototype implementation, and our experience and lessons based on several production applications built with the framework, and outline some continuing work on Paths in the context of the Stanford Interactive Workspaces project.

**Keywords:** Ubiquitous Computing, Automatic Mediation, Service Composition, Ad-hoc Applications, Software Infrastructure

## 1 Introduction

Ubiquitous computing is about enabling people to more easily accomplish tasks through the seamless interworking of the physical environment and a computing infrastructure [14]. This "seamless interworking" implies some level of cooperation among the various devices and software in the environment.

Today, however, most of these devices and software cannot cooperate with each other, simply because they cannot communicate with each other. This stems from a variety of technical causes, but the fundamental cause is that these devices and software simply were *not designed to communicate* with each other. The devices and software were built at different times, for different purposes, and have different capabilities. These problems are especially pronounced when trying to integrate commercial-off-the-shelf (COTS) components and legacy systems into a ubiquitous computing system.

We have two goals: first, to provide a system for connecting heterogeneous, COTS and legacy devices and software, and second, to allow new devices and software to be integrated into the system easily. Our approach is to step away from the tightly-coupled model of direct communication and instead enable loosely-coupled communication through a third party. In this model, two endpoints communicate through a mediating infrastructure rather than communicating with each other directly. This infrastructure *automatically* discovers and places relevant mediators between the two endpoints to compensate for communication mismatches. A loosely-coupled model of communication enables communication between devices and software that were not originally designed to communicate with each other.

## 1.1 For Example ...

An *interactive workspace* is an emerging type of ubiquitous-computing environment. Such a space typically contains a number of highly heterogenous, multi-modal I/O devices: large screen displays, smart whiteboards, projectors, speakers, microphones, wireless pointing devices, etc. In addition, participants in a meeting or other collaborative activity in such a space bring with them a variety of personal devices: PDA's, laptops, cellphones, active badges, etc.

Consider a meeting held in an interactive workspace. As the meeting begins, the meeting organizer displays an agenda (stored on his own laptop) on one of the large screens. During this meeting, the attendees discuss various important points and make decisions, sharing information to clarify and elucidate their respective points of view.

A few attendees have prepared notes and information to share. This information might be in the form of text, graphics or visualizations, spreadsheet data, etc. They pass out electronic copies to the other attendees, who view the information with whatever devices and software they have at hand. Other attendees share unprepared, impromptu data. During discussions, they find or are reminded of relevant documents or elucidating information. Some attendees create documents during the meeting itself. They author these documents using whatever device/software at hand is most appropriate; perhaps a smart whiteboard, perhaps a plain piece of paper (using a scanner to import it into the digital environment).

Looking at this scenario, we see it requires two properties of ubiquitous computing environments:

- Support for Heterogeneous Entities: A ubiquitous computing environment has to support a wide variety of devices, from smart whiteboards to active badges. The environment also has to support not just "permanent" devices, but PDAs and other personal equipment that people bring with them. Additionally, when these entities are legacy or COTS products, or simply not under the environment's administrative control, they cannot be adapted to the environment. The environment must adapt itself to the entity.
- Ad-Hoc Communication Between Entities: None of the devices in this scenario were useful individually. Their purpose was to create and disseminate information. They had to communicate with each other to be useful. Though this scenario explicitly required communication between the entities in the system, the principle holds true in ubiquitous computing systems in general: "No computer is an island" [5].

The difficulty with supporting ad-hoc, any-to-any communication in this environment is the communication mismatches between the heterogeneous entities. Any two entities are likely to speak different communication protocols, understand different data formats, and have different user interaction models. Compensating for all of these differences using existing techniques is not feasible.

## 1.2 Existing Approaches for Communication

Ockerbloom has contributed a very useful analysis of the problem of converting data among various formats [6]. Some of the approaches being used today to mitigate this problem are:

- Standards: one common approach to solving the problem of incompatibility is to create standard protocols, data formats and behaviors for interacting entities. Though standards are useful, we argue that it is infeasible in ubiquitous computing, where potentially all devices and software must be able to cooperate/interoperate. Additionally, standards alone cannot address the problem of nonstandard legacy systems.
- Content Negotiation: a more flexible approach is to add a content-negotiation phase to the communications protocol. However, content negotiation presupposes widely-adopted standards for both data formats and negotiation protocols, and also places a burden on devices to understand multiple sorts of data (not always the case with small devices).
- Polyglot Entities: today, many entities are built understanding multiple datatypes and protocols. These entities can communicate with a larger number of other devices and software, but are still fundamentally limited in their ability to communicate with unknown systems.
- Least Common Denominator Data Formats: using a least-common-denominator datatype and protocol, such as ASCII or HTML over HTTP, often causes a loss of important information that can not be represented in this simple form.

None of these existing solutions satisfactorily addresses the problem of legacy devices and software, or of incompatible COTS products. Nor do they allow

arbitrary ad-hoc communication between two devices: both devices must have been built a priori speaking the same protocols.

## 2 Paths: A Mediation Infrastructure

To address these problems, we have designed and implemented a prototype mediation infrastructure as a part of *Paths*, a general framework for composing mediators distributed across a network of machines. Using this prototype, we have implemented several applications to support improved integration of devices and software, including legacy and COTS entities, in a ubiquitous computing environment.

The mediation infrastructure consists of a set of mediators that transform data, a set of representatives that speak the native protocols of endpoints, and a coordinator that discovers and initializes paths of mediators to connect endpoints. Mediators can be added to the infrastructure dynamically, simply by announcing their existence to the coordinator. Representatives can be added to the infrastructure in a similar fashion.
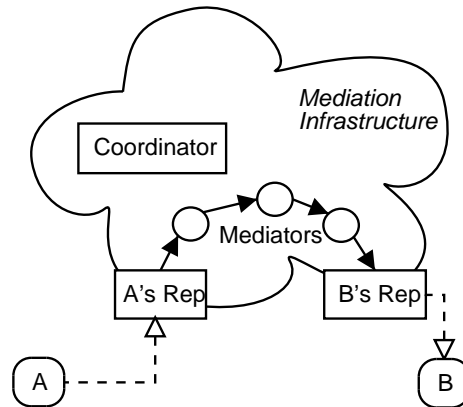


**Fig. 1.** The Mediation Infrastructure

Communication through a mediating infrastructure avoids the problems associated with existing approaches detailed in Sect. 1.2. There are two prerequisites for entities to communicate through this infrastructure:

- There must exist representatives in the infrastructure capable of communicating directly with each of the endpoints. This representative speaks an endpoint's native communications protocol, and forwards data between the endpoint and any mediators.
- There must exist some series of mediators in the infrastructure, which together can transform data from its source format to the format required by the destination. These mediators bridge the data format mismatches between two entities.

Both of these prerequisites are requirements placed on the infrastructure and not on the communication endpoints. This means that after adding the relevant representatives and mediators to the infrastructure, it is possible for existing devices and software to communicate with each other. This system requires no modification of the endpoints, and enables a third party to "wrap" them into a ubiquitous computing environment.

In addition, the mediation infrastructure also makes it possible to integrate new devices into the environment with only incremental work. Instead of needing a bridge between every existing device and the new device, we only need a bridge between some data format already understood in the mediation infrastructure and the new device. Once we have done this, existing mediators will be able to handle converting this data to formats understood by existing devices.

## 2.1 Mediators

Mediators are infrastructure services that transform data from one datatype to another. They generally have a single input and a single output. Datatype descriptions are associated with both their input and output. The fundamental semantics of their functionality are completely described by the strongly-typed interfaces of their inputs and outputs. These mediators may be full-fledged autonomous services, or small pieces of mobile code instantiated on demand.

We can connect two mediators together, running them in sequence, if the datatype of one mediator's output matches the datatype of the second mediator's input. By composing mediators one after the other (into a path) we can create more complex transformations. For example, we can chain a mediator which transforms text into a GIF file with a mediator which transforms a GIF file into a JPEG file. These mediator compositions are responsible for bridging the data format mismatches between entities.

## 2.2 Representatives

Representatives are gateway services between the mediation infrastructure and endpoints. They speak an endpoint's native protocol and data format, and forward that communication into the mediation infrastructure. There is a single representative for each kind of endpoint. As a special case, it is possible for a device to be its own representative in the mediation infrastructure. In our architecture, representatives are responsible for bridging protocol mismatches.

Representatives can either initiate communications with an endpoint, accept communication requests from endpoints, or both. When communicating with its representative, an endpoint does not need to have knowledge of the mediation infrastructure, or of the identity of the other end of the communication. For example, a "dumb" device such as a microphone can send an audio stream to a text editor without knowing what happens to the stream after it leaves the microphone. This is the property that enables us to fully integrate legacy devices and COTS products into a ubiquitous computing environment.

### 2.3 Setting up Communication and Mediation

There are two phases to setting up a Path: determining which two endpoints to connect and determining a sequence of available mediators that bridges the mismatches between the endpoints.

**Choosing the Endpoints** One general issue of communications in a ubiquitous computing environment is "who talks to whom?" That is, which endpoints should be connected together? Our mediating infrastructure does not make this decision itself, nor does it dictate which entity makes this decision. Therefore, we have adopted a flexible control model: a separate controller entity decides which endpoints in the environment should communicate with each other, and notifies the appropriate representatives and the coordinator to establish the communication (as a special case, the controller may be the end-points or their representatives). This decision is made in some out-of-band manner, such as in response to a request by a person, or some set of conditions in the environment. For example, the communications between various applications and displays in an interactive workspace could be controlled using a separate user interface, rather than through the applications or displays themselves.

**Automatic Composition of Mediators** Usually no single mediator provides the exact transformation required to connect two endpoints. In this situation, we must compose multiple mediators in sequence to provide the transformation. In the general case, a human must decide how to compose this mediation. However, when the requirements the endpoints have on their inputs and outputs are completely described in terms of their datatypes, we can automate the composition process. By simply finding a path of mediators that transform from data of the type produced by the source to that required by the destination, we will have bridged the datatype mismatch between the endpoints.

The process of finding a path of mediators can be conceptualized as a graph search. The vertices of the graph are the datatypes. Mediators are the edges connecting the datatypes they transform between. Automatic composition of mediators is thus reduced to finding a least-cost path between two vertices in a graph.[1].

The effectiveness of automatic composition depends in turn on how effective the type system is at describing the assumptions endpoints make on the data they receive. We discuss some of the work we have done to extend traditional type systems to better support automatic composition in Sect. 5.2.

## 3 Paths Prototype

We have implemented this mediation infrastructure as part of Paths, a general composition framework for autonomous services in a network of machines.

A Path is a pipe/filter stream through a graph of operators and connectors. Operators perform computations on data. Connectors transport data between

---

[1] Unfortunately, the implementation of automatic composition is more complicated, due to the existence of parametric types and polymorphic operators.

machines. Operators and connectors interface with one another via queues. Data packets are packaged in the form of Application Data Units (ADUs). An ADU is the smallest unit of data independently processable by an operator [1].

Advantages of using stream-based computing, instead of an RPC interface, include the ability to support unbounded data, such as real-time audio and video, and the potential to provide progressive and incremental computation on large datasets [13]. For example, lengthy speech recognition processes can begin executing before the audio stream has finished.

We use XML-based description languages to describe datatypes, operators, connectors, and paths. The Paths prototype is written in Java, but the entities connected by Paths need not be in Java—the only requirement is that it must be possible to write a Java-based representative that can communicate with the entity. In Sect. 4 we describe several specific representatives we have built to connect legacy applications to our interactive workspace.

### 3.1 Operators and Connectors

An operator consists of a piece of code that performs some transformation and an XML description of its input and output types. An operator may have zero or more inputs and outputs (for example, a data source has zero inputs but one or more outputs). These inputs and outputs are strongly typed and govern how the operator can be composed with other operators. The operator's XML description also includes information on where to get the code (e.g., a URL) and how to run it.

Operators play a number of roles:

- Mediators perform datatype transformations, such as GIF to JPEG or XML to HTML conversions. These operators are the mediators in our mediation infrastructure.
- Semantic processors perform some operation on data that does not change the type of the data, e.g., mathematical computation, sorting or filtering. The dividing line between mediators and semantic processors depends on the descriptiveness of the type system. Semantic operators provide higher-level functionality not represented within our mediation infrastructure.
- Aggregators and Disseminators perform fan-in and fan-out functions within a path. Since our mediation infrastructure currently supports only pairwise communication, it does not use aggregators and disseminators. In the future, they will be used to provide support for treating multiple endpoints as a single, virtual endpoint (e.g., combining two displays to create a larger viewing area).
- Data sources and data sinks are operators which have only a single output or input. From the viewpoint of a path, these operators are generating and consuming data, respectively. In the mediation infrastructure, these data sources and data sinks are the representatives of the true endpoints of the communication, the devices and software.

Connectors, unlike operators, are type-neutral pieces of code described by their transport characteristics. Some of the characteristics that might describe a connector include its reliability, latency, in-order delivery, QoS, and security levels. By default, we currently use a reliable ordered bytestream connector, implemented using TCP. In the future, we plan to allow the controller entity in our mediation infrastructure to decide what sorts of connectors to use based on the purpose of the communications channel; for example, a path manipulating real-time streaming data might prefer an unreliable ordered connector, to avoid retransmissions that violate real-time constraints.

## 3.2 Path Creation

Path creation is the process of turning a logical path description into an instantiation of a path. The first phase of path creation handles assigning operators onto physical hosts and adding any necessary connectors. The second phase of path creation dispatches the path description to appropriate hosts. Finally, the last phase of path creation instantiates the operators and connectors. After this, the path is ready to accept data.

The process of creating a path is itself implemented as a path, with each of the phases of path creation being encapsulated within an operator. The advantage of implementing the path creation as a path is the flexibility we have to add new functionality to the path creation process.
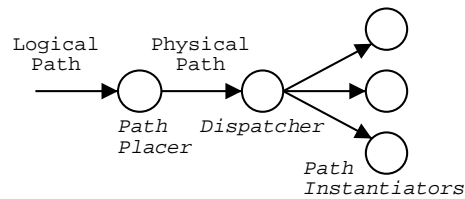


**Fig. 2.** The Path Creation Path

The first extension we have added to the path creation process is Automatic Path Creation (APC), by prepending an extra operator to the beginning of the path creation path. APC is a generic term for taking a high-level query and generating a path from it. The sort of APC that we have implemented is a Partial-Path APC. A partial path is a path description containing mismatches between adjacent operators. Partial-Path APC takes this description and generates a logical path by mediating between the mismatched operators. This logical path is then turned into an instantiation of a path via the original path creation process.

Our automatic mediation problem in the mediation infrastructure is a special case of Partial-Path APC. The generation of the partial path query is the rendezvous problem discussed in Sect. 2.3. Once we have determined which representatives to connect, we generate a partial path query consisting of two op-

erators connected together. These operators model the source and destination representatives, with the appropriate datatype specifications on their inputs and outputs. Applying APC to this partial path query discovers a mediation path between the two representatives.
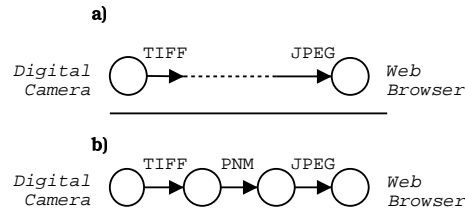


**Fig. 3.** a) a partial-path query between a digital camera and a web browser; b) the result of the query, a full path between the digital camera and a web browser.

## 4 Applications

We have developed a number of services using the Paths prototype and Automatic Mediation. Here, we present three applications with particular relevance to ubiquitous computing, and the lessons we learned while developing them.

### 4.1 Room Control Service

As an initial exploration of Automatic Mediation, we implemented an any-to-any messaging service that enabled users to send and receive messages from potentially any networked device. On top of this system, we implemented a multimodal Room Control service for audio/visual and other resources in several classrooms and conference rooms in Soda Hall, UC Berkeley's Computer Science building. The end devices we connected to this system include a desktop computer GUI, microphone and speakers, a cell phone, and a Palm Pilot.

Users can use graphical, text, or speech based user interfaces to send messages to one another and to programmatic entities (such as the Room Control). When users send a message, it is routed through an automatically generated path of mediators to transform the data from its original data format to the format required by the receiver. These mediators include straightforward data format conversion mediators (GSM audio to PCM audio) as well as more complicated mediators such as speech-to-text and a rule-based text command interpreter. The latter two are interesting because they may inject context-based semantics into the transformation procedure. In Sect. 5.1 we discuss a classification of mediators that has emerged from our experience using Paths to build several applications.

This system was implemented as part of the ICEBERG and Ninja projects at U.C. Berkeley [12, 4]. It leveraged existing software, such as the speech recognizer and IP telephony infrastructure, and was completed in only a few person-months
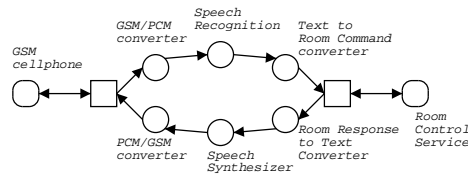
**Fig. 4.** Communication between a cellphone and the Room Control Service.

of work. Extending the system has been equally easy. For example, adding support for sending messages from the Palm Pilot required only two hours of work.

In the process of building this system, we made significant advances in our understanding of paths and our requirements of them. We found that support for parametric types and polymorphic operators (described in Sect. 5.2) greatly increases operator reusability and flexibility.

### 4.2 Dada (Data–Application and Device–Application decoupling)

In the Dada project, we explored the use of the Paths framework for flexible data representation, manipulation and display in interactive workspaces. The goal of Dada was to decouple data/application and device/application dependencies, and give the user the ability to display and manipulate data in an ad-hoc fashion. In particular, Dada allows a user to select what data to view, what semantic processors to apply to this data, and on which device to display the results. Once the user makes these choices, Dada initializes a Path from the appropriate data source, through the chosen semantic processors, and to the display device, using automatic mediation to resolve mismatches between the chosen services. As the data processing completes, the user sees the results on the chosen display, and, if applicable, can manipulate and interact with these results.

We chose to work within the domain of architectural design and construction planning because this domain requires multiple integrated views of and interactions with project data from a variety of sources. The Stanford Interactive Workspaces Project provided the environment and infrastructure for the implementation of Dada [10]. The data sources included 3D models of buildings, time schedules and cost estimates for construction of these buildings, encoded in various XML-compliant markup languages and stored in databases. These data were interlinked, with parts of buildings, scheduled work tasks, and cost items being associated with each other. The device types we supported were large wall-mounted displays and a table display, all capable of showing 2D and 3D data. The table display required its contents to be rotatable so that data may be read from any side of the table.

Some of the more interesting semantic processors we wrote for Dada implemented application-level functionality within the Path itself. These semantic processors manipulated the data flowing through them in response to user actions. These user actions were propagated to the semantic processors using control messages flowing along the reverse path. To ensure decoupling of semantic

processors and the devices displaying actions to the user, the semantic processor packages the control message associated with a user action. In this way, the control message is acting as a callback function. Additionally, downstream operators can transform and even replace these user actions as they are sent to the device, then "undo" this transformation as the control messages flow back up the path.

One example of such a semantic processor we wrote was a data-agnostic operator for Cut&Paste functionality. These operators associate Cut&Paste actions with data objects as they flow through them. Upstream in the path, a mediator renders these associated actions into a device-specific UI description language. Thus, users see live data that they can manipulate and move between displays.

Another interesting mediator converted data for display on the table. It transformed 3D data into rotatable data by wrapping it in a simple user interface for controlling rotation. When the user triggered a rotation action, this mediator applied the appropriate rotation.

Because these semantic processors are both data-agnostic and application-agnostic, it was easy to integrate them into the transformational path. From building the Dada prototype, we gained valuable experience in using Paths and mediators to render user interfaces and dynamic data. Specifically, we gained insights into the requirements dynamic data and user actions had on control messages.

## 4.3   Global Clipboard

The Global Clipboard is a multi-machine, cross-platform clipboard with history. Its purpose is to support transfer of data between heterogeneous, *legacy* applications and devices within an interactive workspace by automatically transforming data among appropriate formats. Target platforms include MS Windows applications and Palm Pilot devices, among others. Target application domains for mediators include image and text document transformations. The prototype global clipboard is currently under development.

Users copy data into the global clipboard just as they would copy data into the normal system clipboard (e.g., by pressing Control-C). A daemon running on each participating machine intercepts the copy operation and forwards a copy of the data to our global clipboard system. When a user pastes data into an application, our daemon again intercepts the request, adds context information about the requesting application and the datatypes that application can accept, and forwards it to the global clipboard. The global clipboard fulfills the paste request by transforming the data currently in the clipboard into one of the formats understood by the application.

By interposing our mediation layer in the clipboard, we are extending an already-familiar metaphor by enabling users to transfer data between applications that otherwise could not understand each other's data formats at all; for example, for cutting and pasting between Microsoft Word and a Palm Pilot memo directly on the Palm Pilot device.

# 5 Discussion

In Sect. 4, we presented three applications that use the Paths mediation infrastructure. Here we discuss some of the lessons we learned about our mediation infrastructure while implementing those prototypes.

## 5.1 Classification of Mediators

Through our experience writing and using mediators in applications, we have discovered a classification of mediators based on the effect they have on the semantic richness of the data they transform.

- Negative-Delta Mediators perform lossy transformations which lose data, structure, and/or semantic information. Examples of this sort of transformation include converting an image from a lossless to a lossy encoding; and transforming XML into HTML, losing the semantic meaning of the original data. They are similar to prior work at lossy encodings to adapt Internet content to small devices [2].
- Zero-Delta Mediators perform non-lossy transformations between equivalent formats. This includes converting an image between two lossless encodings, and converting from a less expressive encoding to a more expressive one (e.g., converting ASCII text to Rich Text Format).
- Positive-Delta Mediators perform a type transformation where some semantic knowledge (context information, etc.) is required to perform the transformation. These mediators are effectively injecting semantic information from domain- or context-specific knowledge into the data as they transform it. For example, Dey and Abowd's CyberDesk [7] uses positive-delta mediation to attach semantic properties such as "this is an email address" to strings that otherwise have no semantics of their own.

This classification of mediators is useful in approximating the degree of information loss that occurs during a transformation. Each negative-delta mediator causes some information to be lost. Positive-delta mediators can re-inject information. However, there is no guarantee that they will re-inject the *same* information that was lost by a negative-delta mediator. Therefore, to guarantee that the existing semantic information is perfectly preserved through a transformation path, all mediators in the path must be zero-delta or positive-delta mediators. In general, the acceptability of negative-delta mediators in a transformation depends on how the transformed data is to be used (e.g., people viewing data can often tolerate more semantic loss than a computer program can).

## 5.2 Extending the Type System

As we noted in Sect. 2.3, the descriptiveness of the type system in our mediation infrastructure has a great effect on the usability of automatic composition of mediators. In order to better describe the data in our systems, we have built a simple, extended type system. Here are the three extensions we made to better describe our data:

- Attributes are <key,value> pairs representing some specific metainformation. For example, our speech recognizer requires that its audio input is not only in PCM sound format, but that is also sampled at 8000Hz. Therefore, the type description of its input includes the attribute <samplerate, 8000Hz>.
- Type Relationships, such as a "contains" or "represents" relationship, describe relationships between types that are awkward to capture with simple subtypes or "is-a" relationships. For example, an aggregator that generates a list of images has an output datatype of "List" with the addition of a "contains" relationship to the "image" datatype.
- WILDCARD and UNKNOWN values are used when an attribute or relationship field does not yet or cannot have a specific value. An attribute or relationship value of "wildcard" means the associated field will match whatever is required. An "unknown" value means no assumptions or assertions can be made about the value of the field.

These data type descriptions are made in a language-neutral format. The type requirements of endpoints are made external to the endpoint. This means we can describe the requirements of third-party entities without modifying them.

It is important to highlight that there is a tradeoff between how descriptive and complete a type system is, and how unwieldy and difficult the type system is to use. As more semantic information is encoded in the type system, the potential for "false negatives" when comparing two datatypes increases. As part of our current research, we are investigating the nature of this tradeoff.

We are continuing to experiment with our extended type system to evaluate its efficacy. Other type systems that may provide enough descriptive power for our purposes include Spreitzer and Begel's Flexible Types [9].

## 5.3 Path Lifetime

In our applications, we have experimented with both long-lived and short-lived paths. Short-lived paths are simple and straightforward, and are appropriate for sessionless message-based communication (i.e., no expectation that multiple messages will follow). Both the Room Control Service and the Global Clipboard use short-lived paths to perform their communication and transformation. The main disadvantage of using a short-lived path is the cost of tearing down and setting up a new path for every message sent.

Long-lived paths can handle a wider variety of applications than short-lived paths. They are more appropriate for session-oriented applications, including streaming media applications and applications that maintain other session state. An example of such an operator might be a "moving-average" operator that tracked the value of a variable throughout the duration of a session. We used long-lived paths in Dada to support the long-lived user interaction sessions.

## 5.4 User Interaction

Our mediation infrastructure provides a progammatic interface to ease development of cross-device applications and ad-hoc applications, but does not enforce

any particular interaction mechanisms upon end users. Instead, Paths leaves the presentation of a user interface to a higher-level task-aware entity.

In our prototype applications, we have just begun to study the problem of user interaction with groups of heterogeneous devices. Our Room Control service built application-aware client programs on each of our supported device which provided the user with a high-level of control. In Dada, we took advantage of general-purpose viewers to generate and render simple user interfaces directly on the client devices. In the Global Clipboard, we hid most of the user interface behind an existing interface, the native system clipboard. In each of these cases, we chose to implement significantly different user interaction mechanisms, but were able to use the same Paths mechanism.

## 6 Related Work

Our work was first motivated in the context of composing Internet services, particularly by work in using web proxies to adapt to client variations [2]. We are still actively working in this area, in collaboration with the Ninja and Iceberg projects at UC Berkeley.

The Compose Group at Carnegie Mellon University has done much work in the area of component composition in the context of software architectures [3, 8]. With respect to their terminology, Paths matches the "pipe/filter model" of software architecture.

Sun's combined effort with Java and Jini is also trying to address the problem of communication in a ubiquitous computing environment [11]. However, they are applying the standards-based approach, and defining a homogeneous system to which entities must conform. We have explicitly attempted to enable communication among COTS and legacy applications and devices.

In the area of ubiquitous computing, our work is most closely related to the context-aware systems designed by the Future Computing Environments group at Georgia Institute of Technology [7]. The type converters or context interpreters described in the context of CyberDesk are the equivalent of our mediators. That work uses type converters primarily to generate context information that can be used as a trigger. We see our work as complementary to theirs and can envision ways in which both systems might be enriched if they could be integrated. Since integration with other systems is one of primary goals, we do not expect this to be particularly difficult.

## 7 Conclusions

Ubiquitous computing requires a seamless interworking of heterogeneous entities, including legacy entities that is not feasible to implement using current technologies.

The mediation infrastructure described in this paper enables ad-hoc, loosely-coupled communication between heterogeneous, potentially legacy and COTS, entities that were not originally designed to communicate with each other. It compensates for mismatches between communicants through the automatic composition of mediators. Due to the use of composable mediators, support for new

entities can be added with only incremental work. We have implemented a prototype of this infrastructure and built a number of applications using it.

Continuing work on the mediation infrastructure includes research into the tradeoffs related to the descriptiveness of the type system, and the use of this mediation infrastructure for supporting dynamically transformed or generated user interfaces, and multimodal interfaces.

## 8  Acknowledgements

## References

1. Clark, D., and Tennenhouse, D. Architectural Considerations for a New Generation of Protocols. *Proceedings of ACM SIGCOMM '90*, Sept. 1990, pp. 201-208.
2. Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *IEEE Personal Communications (invited submission)*, Aug 1998. Special issue on adapting to network and client variability.
3. David Garlan, Robert Allen, and John Ockerbloom. Architectural Mismatch, or, Why it's hard to build systems out of existing parts. *Proceedings of the 17th International Conference on Software Engineering*, April 1995.
4. Steve Gribble, Matt Welsh, Eric A. Brewer, and David Culler. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Aug 1999.
5. Andrew C. Huang, Benjamin C. Ling, John Barton, and Armando Fox. Running the Web Backwards: Appliance Data Services. *WWW-9*, Amsterdam, May 2000.
6. John Ockerbloom. Mediating Among Diverse Data Formats. *PhD Thesis, Carnegie Mellon University*, Jan 1998.
7. Daniel Salber, Anind K. Dey and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. *In the Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, Pittsburgh, PA, May 15-20, 1999. pp. 434-441.
8. M. Shaw, R. DeLine, V. Klein, T.L. Ross, D.M. Young, G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering, Vol. 21, No 4*, April 95.
9. Mike Spreitzer and Andrew Begel. More Flexible Data Types. *In Proceedings of The Eighth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1999.
10. Stanford Interactive Workspaces Project. http://graphics.stanford.edu/projects/iwork/

11. Sun     Microsystems.     Jini     Connection     Technology     Overview.     *whitepaper.*
    *http://www.sun.com/jini/overview/overview.ps*
12. Helen J. Wang, Bhaskaran Raman, et al. ICEBERG: An Internet-core Network
    Architecture for Integrated Communications. *Submitted to IEEE Personal Com-*
    *munications.*
13. J. A. Watlington and V. M. Bove, Jr. Stream-Based Computing and Future Tele-
    vision. *Proc. 137th SMPTE Technical Conference*, pp. 69-79, 1995.
14. Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–
    104, September 1991