xLink: Context Management Solution for Commodity Ubiquitous Computing Environments

Sajid H. Sadi and Pattie Maes

***Abstract*—In this position paper, we present *xLink*, a system for managing context information and marshalling service queries and data flow. The repeated re-invention of context management systems remains a perennial phenomenon in context aware and ubiquitous computing. Often, these systems are integrated into applications to accommodate the low level requirements of the underlying devices. We hope to address this issue by providing a framework that allows a clean, API-based separation of device, context, and service management functionalities without curtailing the capabilities of any segment. *xLink* is designed to allow commodity ubiquitous computing devices, particularly devices with highly constrained computational capabilities, to interface directly at a low level to the context management framework, while still providing a clean and versatile service Application Programming Interface (API) that allows applications maximum expressive freedom. We also present a usage scenario for technologies based on *xLink*, and discuss the strengths and shortcomings of our approach in relation to other context and information flow management strategies.**

***Index Terms*—context management, context modeling, ubiquitous computing, interoperability.**

I. Introduction

Ubiquitous computing (Ubicomp) presents a compelling vision of environments, objects, and even people augmented with technological and computational resources for providing information "just-in-time", or when and where the information is needed and best applied [1, 11]. In introducing this area of investigation, Mark Weiser of Xerox Parc also suggested some critical areas in need of investigation. One of the primary technological challenges noted by Weiser is the *context management* problem, that is, the need for methods of managing and coordinating the flow of information between the nodes and services available in the augmented environment [11]. The omnipresent interface envisioned by Ubicomp requires that devices act concertedly to support the user's current actions [3], and context management forms the basis of this capability. Weiser also noted that the acceptance of the vision of Ubicomp depended, perhaps more than other areas, on reaching a critical mass, or in other words, on the technology becoming commodity. By "commodity technology" we mean that the technologies are commonly available at the consumer level, much as cellular phones or televisions are today. Additionally, we imply high accessibility by the general public, and low initial and recurring cost, such that individuals may use them not as luxuries, but as part of their daily lives. This implies that devices must be low cost, interchangeable, and easily duplicated, manufactured, and used. These constraints in turn bound what can be expected of these devices in terms of processing power, memory, reliability, and adherence to protocols.

In looking at the history of ubiquitous computing, it is clear that researchers have often been forced to develop special purpose context management systems to address their needs [1, 2, 6, 11]. In many instances, the low level integration of the application to the physical devices has essentially led to applications which have context management embedded, thus limiting the applicability as a commodity technology. We believe that this situation is a side effect of the fact that no uniform and expressive API exists to provide a clean separation between device, service, and information flow management.

In this paper, we present an implementation of this claim: *xLink*, a system of context management that offers a general purpose environment which is capable of handling the needs of special purpose, low-capability devices without sacrificing a uniform service model. *xLink* is designed to support both continuously-running and on-demand applications in a seamless way, thus allowing application development independent of the context and device management models. The system helps bridge the gap between general purpose data flow management systems and low level "drivers" that allow simple devices to interact with such systems. *xLink* is designed to be "close to the hardware," and provides APIs for low level interfaces to be built directly above the flexible information routing core. This feature allows us to integrate commodity hardware directly into the context management backend without intermediary layers, while allowing services to interact uniformly with devices without sacrificing low-level access. Finally, the *xLink* system incorporates implicit and explicit invocation of services, which allows for autonomous behavior

Manuscript received June 14, 2005.

S. Sadi is a graduate student at the MIT Media Lab, Cambridge, MA 02142 USA (phone: 617.253.9706; e-mail: sajid@ media.mit.edu).

P. Maes is an associate professor with the Media Arts and Sciences Program, MIT Media Lab, MIT, Cambridge, MA 02142 USA (e-mail: pattie@media.mit.edu).

while still allowing the user control over the interaction and information flow. We have designed *xLink* to be a “room operating system” — a context management solution which can be configured to support the specific purposes of an area, allowing shared infrastructure between disparate services, while retaining the freedom of design and expressivity required by particular applications.

II. Related Work

As mentioned previously, there is a great deal of fragmentation in the area of context management, with many systems existing only as parts of applications. This section provides a non-comprehensive sampling of stand-alone context and device management systems in order to present and contrast some other approaches to context management.

A. *Universal Plug and Play*

Universal Plug and Play (UPnP) is a discovery-based protocol for connecting devices originally developed by an industrial consortium led by Microsoft Corporation. It is meant for spontaneous discovery and configuration of network-connected devices [10]. The communication protocol uses IP as a transport layer, which is generally a complex and resource-intensive protocol to implement with minimal hardware. Additionally, message passing and discovery is done with XML messages [10], which provides high generality but also high latency and resource cost. As such, it is more appropriately considered for sharing pieces of context, as opposed to a system for actual management of context, since devices retain their own context information. While meant for the commodity market, it is too resource intensive for the lowest level devices that the *xLink* project is designed for. Additionally, the peer-to-peer structure, in contrast to the star topology of *xLink*, makes implementation of flow policies highly client-dependent. However, it remains well suited for interfacing devices with greater computational capacity.

B. *JINI Extensible Remote Invocation System*

JINI, an analogue to UPnP created by Sun Microsystems for the Java platform, is also a discovery-based service [10]. Unlike UPnP, JINI does use a lookup service to mediate the discovery process, and the actual discovery involves finding the service in the first place. All subsequent calls are made via Remote Method Invocation (RMI) and serialized Java objects, which is understandable given the tight platform bindings. Due to the requirements of the Java Virtual Machine, JINI is by definition a relatively demanding protocol-oriented method of managing context. The design is tightly bound to a client-server model of interaction that considerably limits its expressive capability. However, the ability to push code from a server to a client is clearly a very powerful capability enabled by the homogenous execution environment, which allows JINI to support greater low-level interaction between devices than UPnP and to achieve some of the low-level interface capabilities of *xLink*.

C. *The InConcert Middleware Layer*

The InConcert middleware layer was produced by the Microsoft Research Labs to serve, among other things, as a middleware layer for the EasyLiving Project [2]. It is, by design, highly specialized for automation and control of living spaces. Like UPnP, it is a protocol based system using XML message schemas. As with JINI, a directory node within an augmented physical location is used to register and track all other nodes [2]. As a result, the system is more centralized, with a central unit coordinating inter-device communication, though the actual communication takes place via asynchronous message passing between individual devices [2]. Like *xLink*, InConcert provides high level lateral linkage support, but depends on a relatively “heavy-weight” protocol (XML over IP) that hinders use in low-capability devices without the use of additional hardware and software resources. One interesting feature of InConcert is its internal topological model of the environment [2]. This highly specialized extension improves upon its intended functionality, which is bound to home automation and control, but at the cost of generality. Nonetheless, the intrinsic understanding of device relationships at the lowest level allows for interesting possibilities that are more difficult with the more generic device model of *xLink*.

D. *The Dartmouth “Solar” System*

The “Solar” system is a message simplification scheme based on directed acyclic graphs. The graph is used to route events to graph operators that incrementally reduced messages to a form which can be consumed by applications [3]. Applications use a context-sensitive subscription model to acquire data [3]. One of the primary problems with ubiquitous computing, evident from its earliest [11] days, is the sheer volume of data produced [4]. The Solar system efficiently compresses this volume by applying aggregating services to incrementally refine information. The high data resolution at the leaves [3] can also lead to performance issues, since the structures are currently geared towards a scenario where subscribing application have information pushed to them. Also, the collaboration of various devices, and additionally the sharing of information between services which operate in parallel (lateral linkage within the data flow graph) is ill-defined. In contrast, *xLink* features well-defined lateral linkage between devices and services at the same level in the overall hierarchy, but does not focus on data aggregation, thus allowing easier inter-device and inter-service collaboration capabilities at the cost of scalability.

E. *The EventHeap Model*

The EventHeap model is a middleware layer designed for the Stanford iRoom, and allows the multiple input and output surfaces of the room to function as a ubiquitous computing environment [7]. It is based on an API model, though internally the implementation does use a protocol to communicate between targets. The API model differs from the *xLink* model in that it is meant for programming devices themselves, and not links to devices. This suggests a level of capability with is easily available within the iRoom, but cannot be presumed for the scope of *xLink*. The message passing strategies of both systems are also similar, though unlike *xLink*, EventHeap tends to treat devices and services essentially as peers, since within the EventHeap's scope devices and services are generally intertwined.

III. Scenario and Case Study

In order to motivate the system design, we would first like to present a short scenario. Though this is a fictitious usage scenario, the devices and technologies discussed are the outcome of research projects completed by our group using *xLink* as the context management system.

Scenario — "Rodger walks into the local bookstore, hoping to find a good book on yoga. The bookstore is enabled for his *ReachBand* — a wireless Radio Frequency Identifier (RFID) and gesture sensor worn on the wrist [5] — and headset, and he quickly scans the placard at the entrance to enable his devices. The store had already detected the wristband, and by reading the RFID tag on the placard, Rodger gives explicit permission for the store system to interact with his devices. As he walks around, a book catches his eyes, and he picks it up. As he is flipping though the book, the *ReachBand* scans the tag on the book, and he hears a slight audio notification on his headset that information of that book is available. He finds the book outwardly interesting, so he makes a slight gesture of his wrist, which accesses the list of services. The system has retrieved his profile from his phone, and knows he likes the New York Times reviews when possible, so when he selects reviews, the system checks and tells him that there is a New York Times book review available. He chooses to listen to the review. The review plays over the headset as he continues to flip through the book. He then chooses to listen to the Ambient Semantics [9] service, which tells Rodger that several of his friends are interested in getting into yoga exercises, opening the possibility for working out together. He also notices that a review selected by the system recommends a different book for beginners. Rodger is in a rush, so after finding the other book, he scans the tag on a public access display, which shows the menu he had been listening to. He quickly accesses and scans the review, and chooses to buy the new book. He simply selects the buy option, and authorizes use of the card he has in his profile. He signs the store receipt on-screen, but if he had not been at the terminal, the clerk would simply have had the receipt ready for him at the counter. The security system has been made aware that the book now belongs to Rodger, and automatically disarms the security tag as he walks out."

Within the above scenario, the *xLink* instance for the store has coordinated the experience so that the sensor and output devices that Rodger brought with him, and embedded systems and services within the store work in coordination to produce a seamless experience. The system invokes both local services, such as services to buy books, detect gestures, disarm a security tag, and read back information as well as remote services, such as the New York Times book review search. The system also allows a coordinated privacy policy to be implemented, even though multiple systems are used, so Rodger only has to authorize the system to allow access from a single location. Lastly, the output of the system is seamlessly transformed and routed to output devices, including the headset, the terminal, and the receipt printer at the counter, as needed. Input is also gathered from multiple devices to feed services such as the buy service, which requires an additional signature input. With this scenario in mind, we shall outline the design choices that support this interaction.

IV. System Design

A central goal of the *xLink* system is to provide a local-scale generalized framework for connecting commodity devices. As such, we chose to create a system which is based on an API-based plug-in design instead of protocols, so as to ensure overall generality and "instant-on" capability for new device types. In exchange, the system gained the ability to have highly device-specific behaviors that allow for easy modifications and additions at the interface without requiring changes to the internal workings of the core software. In a protocol based system, for example, the store system would have had to communicate with the wristband via some standard protocol, which would also then have to accommodate all similar devices, or suffer from myriad parallel protocols catering to different systems. Instead, by supporting a pluggable interface, we are able to support a very minimal communication system between the *ReachBand* and *xLink*, while retaining flexibility for the services. As Cohen points out [4], this is an important goal in creating a system which can survive unanticipated requirements, and the API-based interface also supports this easily. If, in the next year, a different model wristband becomes common, it is simple to install another pluggable module and necessary transceiver to support the new device without requiring modifications to the core *xLink* system.

The principal purpose of the system is to mediate interaction between input and output devices, which may physically be instantiated as objects, rooms, or even conventional computer displays, and services. To this end, we designed the internal representation of the input of devices and the output of services to attain flexibility, and focused on allowing a high level of adaptability for interfacing physical devices to the system. This can be seen in the seamless transitions between display and interaction modes, which reflect the flexibility of the internal representation. It is important to note that though it is often handled within the realm of context management, the management of object identity and metadata is implemented as regular services within our system, and is accessed like any other service as needed.

Tuple-Set::ReachBand

name	values	
device	ReachBand	
id	17:32:54:af:7b:92	
payload	64:8c:d4:56:b3:9f:7a:14	1118990279918
gesturedata	73,44,3,35,23,34,23,5,623,23,42,42,34,23, 23,4,23,54,2,52,35,23,3,42,3,42,34,23,42, 3,5,14,56,82,34,82,0,3,57,2,85,90,31,48, 75,12,37,89,567,34,90,57,34,8,9,0,67	

Figure 1: Typical tuple-set from a scanner device in the shared data space after processing by link module.

A. *Internal Representation*

Internally, the system operates on "tuple-sets" – unordered sets of n-tuples, with each tuple consisting of a name as the first entry, and n-1 data values as the remainder. A representative tuple-set is shown in Figure 2. The overall schema is very similar to the system originally used in the Linda system [6] and later explored in the EventHeap system [7]. Each tuple-set has a required field, "id," which identifies the logical device that originated that tuple-set uniquely within the context of the system instance. This accommodation is made so that devices like the wristband in our scenario, with limited communications capabilities, may still identify themselves without having to expend precious bandwidth transmitting a universally unique identifier (UUID). Additionally, the link layer can attach a family identifier to the actual identifier received to partition device families. Therefore, in Figure 2, it might be that the device model itself only received the very last byte from the *ReachBand*, and provided the rest locally. If necessary, the device model can also send commands directly to the core system using tuple-sets. This allows the device model to communicate with the system without incurring additional API overhead.

The communication is generally analogous to the "tuplespace" concept introduced by [6], but using tuple-sets in the same way as EventHeap [7]. Our implementation allows for sets of tuples, without imposing the temporal constraints of EventHeap. Instead, depending on the device type, certain fields are required in each tuple set to identify the source and provide a means of "replying" to a message. The tuplespace methodology depends on matching of key sets against the requisite key sets of a service [7], which is analogous to the service selection system used within *xLink*. Tuples with no consumers, if marked ephemeral, are simply discarded. Devices may economize bandwidth and lower overhead by storing fragments of data streams into fields. In figure 2, the *ReachBand* has stored accelerometer data for gesture detection in this way, sending several records in one tuple. The entire tuple [<id, *device id*>, <gesture-data, *data point1, data point 2, …, data point n*>] can then be consumed by a gesture translation service which converts the data into a single tuple-set [<id, *device id*>, <gesture, *gesture-type*>], retaining the same identifier as the original.

The output of services that is meant for display devices is internally represented as structured trees, which are contractually constrained to provide information in increasing detail ordered by depth. This is a "soft" requirement, which may be ignored if necessary. For example, if there is some minute detail of high interest, it may be placed at a higher level. Additionally, resource-expensive or highly specialized services can opt to send a request link indicating their availability instead of the actual output. This, for example, allows the New York Times search service to reply with links to several articles, without having to send all of them. Links to content rendered in different formats can also be embedded in the tree, which can then be retrieved by devices capable of rendering the link. As a result, images of the book suggested by Ambient Semantics can be displayed by the terminal, but easily ignored by the headset. Within the design constraints of the system, this allows for a reasonable range of output modalities without requiring a highly annotated output structure.

B. *Device Model*

All devices within the scope of the system are classified as logical output or input devices. For example, the headset or the terminal screen are logical output devices in our scenario, while the RFID sensor and the security tag sensor are input devices. Accommodations are made for complex (multi-modal) devices such as the terminal, which are internally represented as a collection of logical input and output devices. Since we expect to have commodity devices from a variety of sources and with varying levels of complexity and capability interact with the system, very few assumptions are made about devices. Each family of devices is allowed its own plug-in "link module," which is responsible for translating data to or from a format the device can accept. Link modules communicate to the devices through shared "connector modules," allowing for sharing of resources among devices using the same communication medium, but differing protocols or information transformation needs. For example, several different phone clients can be supported, all over the same Bluetooth connector module, but via different link modules for different manufacturers. The interaction between various parts of the device model and the *xLink* core context management system is shown in figure 1. As shown in the figure, all connections between logical subsystems are by default bidirectional, allowing modules to be both data producers and consumers. As such, link and connector modules may act as local services, consuming input or carrying out other operations independently as required by the device. For example, the wrist gestures can be consumed directly by a display link module lacking a keyboard, without posting it to the event pool. In addition, this allows for the creation of low-latency links between input and output devices, thus bypassing the internal tuple-set pool for time-critical information.

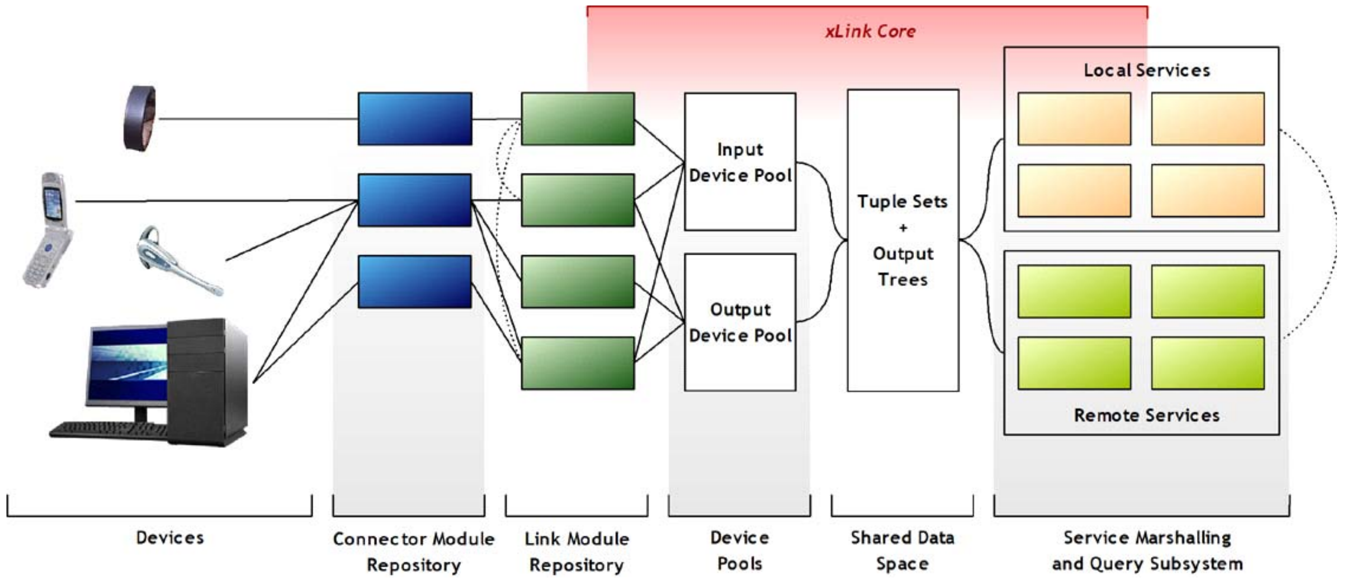

Figure 2. Overview of the *xLink* linkage model, showing interconnections between different module families and object encapsulation. The core *xLink* functionality is indicated by segment marked "*xLink* Core" and is responsible for the message routing and service querying functionality at the heart of *xLink*.

As stated previously, for input devices, the link module simply translates to the tuple-set format. In addition, the link module can perform any transformation on the incoming data, thus providing the ability to augment the actual data sent by the device. In our scenario, this allows the wristband to use a very short ID's and reserve the majority of the packet space for RFID or gesture data. In the case of the display link modules, the wide range of possible presentation resolutions and output modalities require that the module make adaptive decisions based on the context of the interaction to exclude some part of the result tree from the actual output if necessary. This becomes especially necessary when device constraints such as memory or bandwidth make the make of sending the entire dataset literally impossible. The link module may also carry out semantic transformations on the data, ie, linearizing text for a brail display, or converting text to speech for an audio device, and cache the information for the duration of an interaction. These properties are exploited in providing speech output to the headset in our scenario, while stripping out hyperlinks and images in the result set.

As Kindberg [8] points out, it is generally good practice within fault-tolerant systems to presume that devices will fail on a regular basis, and that is the approach taken by our design. Input and output devices are held in separate pools, allowing for both overlaps and multiple entries for the same physical device. Input devices are added to the pool when they make contact, and are removed after a timeout period. This prevents the pool from becoming stagnant (ie, containing logical devices that are no longer communicating). Output devices are likewise added to their own pool on first contact. However, output devices are only discarded if the link or connector module notifies the pool of a failure, or if an associated composite physical device is discarded from either pool due to failure in communication. The connector or link module may additionally force the device to reconnect with some fixed period in order to "clean" the pool. In all cases, output devices are contractually bound to reconnect on loss of connection, should they wish to remain within the network of devices. In our implementation, a garbage collection system discards stale displays after a considerable delay. These devices, if active, then automatically reconnect to the system. This method of connection management is optimized for the expected communication modes of the input and output devices, with inputs "pushing" data to the system, and data being "pushed" to the outputs from the system. As such, no action is required by Rodger to disconnect as he leaves. The loss of Bluetooth connection automatically causes the system to purge his records from the pools.

C. *Service Model*

The system supports the concept of local versus remote, and simple versus complex services. We define local services as services that act on information, objects, and entities within the general locality of the object itself (possibly even on the same object), while remote services act upon objects outside that domain. For example, in our scenario local services provide identifying information about the book, and capabilities such as purchasing, while remote services such as the review search are accessed from the appropriate sites as needed. Likewise, complex services act upon the output of simple services, while simple services act upon local information. For example, transforming acceleration data to gesture identifiers is a simple service, while retrieving the reviews is a complex service that must inspect profile information (possibly a remote complex service itself) to make a decision, and then fetch the appropriate information from a remote service to output. The primary reason for the existence of this distinction is speed of access. Local and simple services help to make an environment "smart" by allowing the various parts of the environment to act in concert. At the same time, most of the decisions made expected to be fast, resource-frugal, and fairly simplistic. As such, the system can call of these services constantly and push their results to output devices without user intervention. The remote or complex services, on the other hand, are "heavyweight" by comparison, and thus their use is limited to on-request accesses.

Since the remote services are only accessed on request, the discovery procedure is reduced to the services notifying a centrally accessible directory service of their existence. Such announcements must of course be accompanied by a list of service and tuple-set dependencies, so that the system can filter out non-relevant services. All local services operate under the prevue of the context management system, thus obviating discovery. As stated previously, the scope of this project allows us to assume that the system has been specialized to the environment it is embedded in. As such, the local service set can be chosen to match the supported devices, and any remote services requiring constant invocation can be proxied by a local forwarding service.

D. System Operation

Being geared towards use in ubiquitous interfaces and smart objects, the system in general “pushes” data to target devices. Incoming tuple-sets are automatically processed by the local simple services and made available to link modules for output. For example, incoming RFID data is automatically disambiguated but the identification service, and the output is routed to all local services by the *xLink* core for processing. Any output is made available to output link modules for display. The link modules have control over whether the data is actually sent, while the services retain control of whether it is appropriate to produce output given a certain scenario. This allows the output devices to exhibit per-device and per-family “do-not-disturb” behavior, such as disabling notification to the headset, while allowing services to operate in a context sensitive way. It is generally understood [2, 8, 7, 11] that ubiquitous interfaces must be proactive in anticipating needs, while at the same time working as a spatial and contextual filter for information so that the user is not inundated by requests for attention. At the same time, ubiquitous interfaces must allow the user control over the interface [1]. The chosen distribution of responsibility for inhibiting input is therefore balanced so that object and devices continue to act like objects (ie, remain under the user’s control), but the overall environment can still retain proactive agency.

While we have not discussed user sessions explicitly to this point, ownership of devices and sessions is a natural extension of the system. We define a session simply as the set of all information available about a user through all available devices and services. When a device is aware of its ownership, it can inform the system of the ownership via a command tuple, which in turn propagates this information to other devices connected to the session. For example, the cellular phone can identify the other devices that a person is carrying that are available for input or output in our scenario. However, ownership is a fluid property within the system, and can change easily. This fluidity is in some sense contrary to the privacy that context management systems must support [1]. Within *xLink*, this factor is mitigated by having devices with known ownership, such as the cellular phone in our example. Services have the option to check for such devices before sending information, or requiring additional authentication if the information is especially sensitive. Due to the low resolution nature of the most common devices, this is expected to be the exceptional case, and therefore security policies are themselves implemented as a service which other services may query.

Like the cell phone or the public terminal, certain devices may have the ability to request additional services by virtue of being composite input-output devices. As a natural side effect of the architecture, a device may easily direct the results of its request (via a command to the system) to a different device. Therefore, the wristband can easily have its output sent to the headset or even to a stationary display designated for that purpose. As long as the device can find out about the target device in some way, or the target device can be selected by the user to be the output device of choice, the output, being naturally device-agnostic, can be routed to it. This allows the system to fulfill the goal of Ubicomp of providing a seamless end-to-end experience to the user without device lock-in during an interaction [11].

V. Discussion and Future Work

The design intentions that guided the *xLink* project were geared towards providing a highly adaptive, seamlessly expressive data flow management system that is general purpose in its architecture, but intended to be customized to support the specific requirements of the applications it serves. As a result, we envision *xLink* as an embedded device, perhaps embedded into a wall of a room enabled for ubiquitous computing. As a part of the room, it can then be tuned to the purpose and interactions that the room is meant to support. As with any design tradeoff, this decision has both its strengths and weaknesses. It is highly adaptive, and capable of hosting a variety of connector and link modules that allow fine grained, low level control of the communication and data translation capabilities offered by an instance of *xLink*. As a result, it is also less capable of self-configuration and management. The highly fault-tolerant design of the system allows for minimal maintenance, but the initial setup cost is not essentially mitigated by this. Additionally, the tuple-set and device pool based internal architecture become limiting factors as the data that must be represented becomes more complex and dependent on factors outside the direct prevue of *xLink*.

Nonetheless, we believe that *xLink* serves to fill an important niche between medium-scale context management solutions such as EventHeap [7] and the devices themselves. By providing a common architecture for the housing and maintenance of driver-like structures that connect devices to each other and to services, *xLink* abstracts away the low level complexity of such information management without impeding the capabilities of the system, much as an operating system hides the complexity of the devices that make up a computer system while presenting software with the ability to access and manipulate those devices. At the same time, *xLink* provides an alternative and parallel for heavier protocol-based systems such as InConcert [2], in that it allows commodity sensors with minimal capabilities to connect to each other and to relevant services, without requiring the overhead of a protocol stack. Finally, the remote service invocation infrastructure of *xLink* allows the system to easily yield to other, more general, context management frameworks such as EventHeap [7] for moderate-scale service invocation and information sharing, and also to use simplifying models such as Solar [3] to filter data down to forms that high-level complex services can use, without having to explicitly support such activities internally.

In continuing our work on *xLink*, we hope to tackle some of the outstanding questions that remain for both this project and for Ubicomp in general. In particular, it appears necessary to support security and privacy primitives within *xLink* itself in order to optimize adherence to privacy and personalization policies. While the current service-based implementation allows for simple adherence to policy, the adherence is only contractually enforced, when it is best enforced directly in the core environment. However, the best way to do so without impacting performance or free flow of data has not yet been fully investigated, and is distinctly in need of further consideration as ubiquitous computing solutions gain in both popularity and resolving power, leading in turn to increased risks to privacy. Additionally, the idea of sessions is currently very loosely implemented within *xLink*, which circumvents several crucial issues with sessions. In particular, the issues of data duplication and disambiguation remain open to further investigation, especially in a highly localized system such as *xLink*. As this work progresses, we hope to engage some of these issues in the design space that is addressed by *xLink*.

Acknowledgment

We would like to acknowledge the tireless help, critique, and encouragement of Assaf Feldman and the Ambient Intelligence Group, as well as the industrial liaisons of the MIT Media Lab for fostering the growth of this project. We also extend our appreciation to Aaron Zinman for the critique, review, and inspiration he provided for this paper.

References

[1] G. D. Abowd and E. D. Mynatt. "Charting Past, Present, and Future Research in Ubiquitous Computing," in ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1, March 2000, Pages 29-58.

[2] B. Brumitt, B. Meyers, J. Krumm, A. Kern and S. A. Shafer. "EasyLiving: Technologies for Intelligent Environments," in Handheld and Ubiquitous Computing, Bristol, UK, September 2000, Pages 12-29.

[3] G. Chen and D. Kotz. "Context Aggregation and Dissemination in Ubiquitous Computing Systems," in Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, 2002.

[4] N. Cohen, A. Purakayastha, J. Turek, L.Wong, and D. Yeh. Challenges in flexible aggregation of pervasive data. IBM Research Division, Thomas J. Watson Research Center. Technical Report RC21942.

[5] A. Feldman, E. M. Tapia, S. Sadi, P. Maes. "ReachMedia: On-the-move interaction with everyday objects," in review for the International Symposium on Wearable Computing, 2005 (ISWC05). **

[6] D. Gelernter. "Generative Communication in Linda," ACM Transactions on Programming Languages and Systems, 1985. Vol. 7, Pages 80-112.

[7] B. Johanson. "Application Coordination Infrastructure for Ubiquitous Computing Rooms." Desertation submitted for Doctor of Philosophy in Electrical Engineering from Stanford University. Available: http://graphics.stanford.edu/~bjohanso/dissertation/johanson-thesis.pdf

[8] T. Kindberg and A. Fox. "System Software for Ubiquitous Computing," in IEEE Pervasive Computing, Vol1, No. 1, 2002, Pages 70-81.

[9] H. Liu and P. Maes. "InterestMap: Harvesting Social Network Profiles for Recommendations," to appear in the proceedings of the Beyond Personalization 2005 Workshop, January 9, 2005. ACM Press 2005.

[10] R. E. McGrath. "Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing," Department of Computer Science University of Illinois Urbana-Champaign, UIUCDCS-R-99-2132, March 25 2000.

[11] M. Weiser. "Ubiquitous Computing," in IEEE Computer "Hot Topics," October, 1993. Available: http://www.ubiq.com/hypertext/weiser/UbicompHotTopics.html

** This paper is currently not available publicly. We would be glad to share a copy if there is interest.

Sajid H. Sadi is currently a first year graduate student at the MIT Media Lab in Cambridge, MA, USA. Prior to joining the Media Lab, he received his BS in computer science from Columbia University in New York, NY, USA in 2003. He is interested in systems design, visualization, interactive design, and techniques for integrating the physical reality with digital metadata.

Pattie Maes is an associate professor in MIT's Program in Media Arts and Sciences. She founded and directs the Media Lab's Ambient Intelligence research group. Previously, she founded and ran the Software Agents group. Prior to joining the Media Lab, Maes was a visiting professor and a research scientist at the MIT Artificial Intelligence Lab. She holds bachelor's and PhD degrees in computer science from the Vrije Universiteit Brussel in Belgium. Her areas of expertise are human-computer interaction, artificial life, artificial intelligence, collective intelligence, and intelligence augmentation. Maes is the editor of three books, and is an editorial board member and reviewer for numerous professional journals and conferences. She has received several awards: Newsweek magazine named her one of the "100 Americans to watch for" in the year 2000; TIME Digital selected her as a member of the Cyber-Elite, the top 50 technological pioneers of the high-tech world; the World Economic Forum honored her with the title "Global Leader for Tomorrow"; Ars Electronica awarded her the 1995 World Wide Web category prize; and in 2000 she was recognized with the "Lifetime Achievement Award" by the Massachusetts Interactive Media Council.