Smart Environments: Middleware Building Blocks for Pervasive Network Computing (A Position Paper)

Jon Weissman and Zhi-Li Zhang

† Dept. of Computer Science & Engineering
University of Minnesota
Minneapolis, MN55455
{jon,zhzhang}@cs.umn.edu

Abstract

With the explosive expansion of the Internet as the global information infrastructure, coupled with rapid advances in wireless technologies, a new vision for computing is emerging. A seamless, anywhere, anytime, untethered, pervasive computing paradigm is touted as the next major paradigm shift in the way we compute and communicate. In order to realize this vision, the complexity of this environment must be masked from its consumers (users) and producers alike (service providers). We believe that a middleware infrastructure that provides appropriate high-level abstractions for both users and service-providers is needed. Such a middleware infrastructure must allow powerful services to be easily deployed and permit users to interact with such services in a personalized, yet effortless manner. In this position paper, we introduce the notion of a "smart service" – services with which users can dynamically interact in an automated and personalized manner, and can be composed to form "smart environments." Smart environments present an *integrated* view of services to the user to enable much *higher-level, domain-specific* interactions and to support user customization. We present a system architecture for building smart services and smart environments, and the middleware research challenges that must be addressed to realize this architecture and the core abstractions.

1 Introduction

With the explosive expansion of the Internet as the global information infrastructure, coupled with rapid advances in wireless technologies, a new vision for computing is emerging. A seamless, anywhere, anytime, untethered, pervasive computing paradigm is touted as the next major paradigm shift in the way we compute and communicate. Towards this ultimate vision, many challenges, technical and otherwise, must be overcome. In order to realize this vision, the complexity of this environment must be masked from its consumers (users) and producers alike (service providers). We believe that a middleware infrastructure for pervasive computing must provide appropriate high-level abstractions for both users and service-providers. Such a middleware infrastructure must allow powerful services to be easily deployed and permit users to interact with such services in a personalized, yet effortless manner. In this position paper, we introduce the notion of a "smart service" – services with which users can dynamically interact in an automated and personalized manner, and can be composed to form "smart environments." Smart environments present an *integrated* view of services to the user to enable much *higher-level, domain-specific* interactions. This integration also facilitates the construction of new smart services and allows the smart environment to serve as a building block for deploying even more powerful and functional environments across both wireless and wired networks. Such a middleware infrastructure will ultimately bridge the gap between human users and technologies, allowing users to utilize information technologies and services unconsciously, just like the way we use today's common utilities such as water and electricity.

We believe that this middleware infrastructure must permit incremental construction and deployment of services, to scale up. For example, we can start with simple smart services such as smart printers, smart email, smart calendars to form a smart desktop or smart workspace; the smart workspace can be extended to more functional environments such as smart offices, smart classrooms; which can be scaled to even larger and more functional environments such as smart buildings, smart departments, smart campuses, etc. In building such environments bottom up, integration is the key: *it calls for a common set of fundamental abstractions and principles that allow smart services and environments to be easily customized (to support users) and extended (to support* both *service-* and *environment-providers).* Integration allows new smart services to be more easily defined and added to an existing smart environment, and allows existing smart environments to be extended to act as building blocks to form larger and more complex environments.

Three key abstractions are the cornerstone of the smart environment (SE):

- a *domain-specific, environment-wide shared state space* which defines the SE semantics, stores and manages SE states and events of common interest to users and services
- *execution context*, a mechanism that provides dynamic management of per-service-invocation state
- a *flexible, declarative API* based on XML for user-service and service-service interactions that permits users to express preferences (*user signature*), allows services to describe their behavior (*service signature*), and allows SE functionality to be easily described and extended (*SE signature*).

In this position paper, we advance several fundamental abstractions and concepts, describe the basic system architecture for building smart services and environments, and present the middleware research challenges that must be addressed to realize this vision.

2 Motivating Scenarios

To help motivate the benefits smart environments can provide, we consider an example – a "smart virtual department". This "smart virtual department" (an integrated smart environment for the Computer Science Department, University of Minnesota) provides customized and personalized services to its users: faculty, students, staff, and visitors, both within the confines of the department and beyond, utilizing in-building wireless networks. Through this example we hope to illustrate how a complex environment that to-date offers little automation, coordination or customization can be potentially transformed into a smart environment with an integrated collection of smart services.

Today our department performs a diverse set of functions: teaching, research, community service, outreach, etc. To support these functions, it manages a variety of resources and events: physical and computing facilities (classrooms, offices, machines, printers, etc.), course materials (lectures, schedules, etc.), research materials (papers, code, etc.), internal and external events (colloquia, interviews, etc.), and so forth. With increased information overload and various other demands on our time and attention, more and more faculty, students, and staff feel overwhelmed and stressed to work in such an environment. The purpose of the smart virtual department is to provide an integrated collection of services that its users can utilize with relative ease and in a seamless, automated, and customized fashion. We consider several scenarios how users can interact with the smart virtual department.

1.a) Professor X has just finished the final edits on his lecture in a powerpoint file on his office machine. When the document is closed, the smart lecture preparation service automatically copies this file to the course web site, along with pdf, ps, and html versions that it generates. This system also schedules a lecture copy event that will copy the powerpoint slides from the office machine to a smart projector in the classroom 20 minutes before lecture on the appropriate day. Furthermore, it automatically notifies students of the availability of the new lecture via a smart email service. While walking to lunch with his PDA before the lecture, the professor remembers that he wanted to print out a set of handouts for the class. While mobile, he asks a smart printer service accessible to his PDA to print the handouts to the "best" printer. The smart printer service locates a printer that is lightly loaded and nearest to the professor and replies with a notification of where the output can be picked up.

1.b) Waking up late in the morning, the first thing Student Y of Professor X's class does is to check his smart calendar. His smart calendar reminds him that the homework for the class is due today, and asks him whether he wants to schedule electronic submission of his homework. Student Y types in the file name of his homework, clicks yes, and then jumps into the shower. On his way to a coffeeshop, Student Y receives a short text message, telling him that the new lecture is available for downloading. Using his cell phone, he instructs his virtual study aid service to retrieve the lecture to his laptop. With a cup of coffee in hand, Student Y goes to the study hall and previews the lecture. The lecture is encoded with meta-information such as FAQs, background material, and links to important relevant concepts introduced in previous lectures. Student Y first reviews these concepts before proceeding to the new lecture. While reading the new lecture, he marks down those new concepts which are not very clear to him.

1.c) As students enter the classroom at the lecture time, new announcements are automatically uploaded to their personalized smart classroom desktop located with each "seat". During the lecture, student Y annotates his copy of the lecture as Professor X goes over it, paying particular attention to those concepts he marked earlier. He occasionally raises his hand and asks Professor X further questions for clarification. Professor X frequently engages the students with interactive Q&A sessions, where students type in answers to questions raised by Professor Y. Using the smart lecture system, Professor Y picks a few typical incorrect answers submitted by the students. He goes through each answer, solicits their opinion asking why an answer is correct or incorrect. Through these interactions, Professor X helps the students gain a better understanding of the new material introduced in the lecture. Meanwhile, student Z is sick in bed and misses the lecture. Fortunately, the classroom is streaming the lecture (not only the video image, but also a trace of the interactive Q&A sessions conducted during the lecture). The lecture is uploaded to the virtual classroom remote access system. Via this virtual classroom remote access system, this student can experience the lecture as the students in the classroom.

2.a) Remotely, an interview candidate wishes to access our smart virtual department system to obtain information about the department. Once given a special key, she can access an array of services including: "sitting in on a lecture," "locating all faculty in her area," etc. Once her itinerary is established, it is sent to the candidate along with a viewable map or triptik showing each step of their interview. Going over the interview schedule, she marks a few faculty members on her schedule to automatically retrieve their research profiles and other information. When she arrives on the interview day, the triptik will be automatically integrated with a department navigation service that tracks her current position and shows how to get to the next office, nearest restroom, etc. An interview monitoring service is also following the course of her interview, and notifies any downstream faculty of delays.

The salient features of the smart virtual department we present above is that it offers an array of integrated services in a specific domain tailored to a wide-range of "consumers." Within the smart virtual department, services are aggregated: the smart virtual department itself consists of many integrated smart services such as smart classrooms, virtual lecture system, virtual student aid, smart printer, smart email, smart calendar, and so on. With its (virtual) presence that extends across the Internet accessible via remote and wireless access, the smart virtual department environment is not limited in its reach by physical location. It is also much more than a "passive website;" the provision of smart services makes the smart virtual department "active." Furthermore, the smart virtual department can also be part of a larger smart environment, e.g. a smart virtual campus, or a virtual national community of computer science departments or virtual organizations that tie together different campuses.

3 Smart Environment: Fundamental Abstractions and Core Architecture

To implement the vision articulated in the preceding section, a middleware infrastructure for building smart services and environments in a coherent, efficient, and scalable manner is needed. We present a common set of fundamental abstractions and concepts for developing a smart service and smart environment middleware infrastructure, and the research challenges that must be addressed.

3.1 What Is a Smart Environment and Why Is It Useful?

A *smart environment* (SE) is an *integrated* environment that allows users and services to interact with each other in a *flexible*, *personalized*, and *domain-specific* manner. We believe that the SE abstraction can be used as a building block for constructing pervasive next-generation computing environments. The "smartness" of an SE allows its users to navigate through the environment with relative ease and in a *personalized* (and possibly *invisible*) manner. We avoid the term "intelligent" to emphasize that we do not assume that the environment has any cognitive power, e.g., the ability to learn, or evolve on its own. In other words, we do not assume that the environment is endowed with "artificial intelligence," although we do not preclude it. The term "environment" is used to indicate that there is an implicit delineation of the basic functions of an SE and its (physical or virtual) boundary. An SE is designed and built to perform specific tasks, i.e., it is *domain-specific*. For example, a smart virtual workspace is designed to provide a location-independent virtual working environment that can be conveniently utilized and managed by users. A smart building helps users of a building easily navigate through it and locate the many amenities it provides.

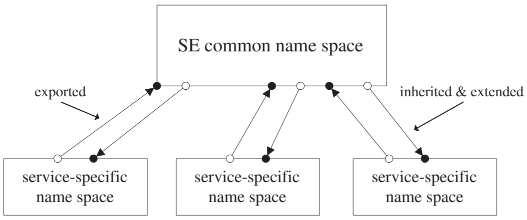
Each SE is provisioned with a set of smart services that collectively perform certain domain-specific tasks. A *smart service* is integrated within the SE fabric to provide customized behavior to its users. Other services may also exist within the SE, but are less integrated and customized. For example, various device proxy's allow services to communicate with heterogeneous devices in the SE, but do not provide customized behavior. Similarly, a bandwidth sensor is a service that provides bandwidth measurements to the SE and offers no customization. Everything in the SE is a service with an API describing its behavior (including users that are represented by user proxy services). However, an SE is not merely a collection of smart services, proxy services, and sensor services that are capable of communicating with each other: an SE is greater than the sum of its parts.

Services within the SE are integrated in a structured manner to present a shared higher-level (SE-level) service abstraction. For example, consider a very simple SE that offers a smart printing service that manages a collection of diverse printers (each running a proxy service that provides dynamic information about its status). A smart printer service is more than a collection of printers. It provides a high-level service abstraction that allows a user to interact with it as a single entity: e.g., a user can request that a document be printed with color as soon as possible; the smart printer service would automatically choose a color printer that has the lightest load. An alternative solution would have the user query each printer and select an appropriate one. Interacting using this service-level abstraction (individual printer proxies), as opposed to a SE-level abstraction (smart printer service), would undoubtedly place more burden on the user. We believe that such domain-specific high-level service abstraction is critical to building scalable SE's in an aggregated manner.

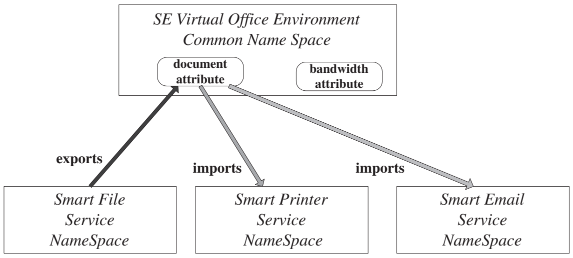
Primitive SE's that offer simple functions (e.g. the printer SE) can be built from scratch. Simpler SE's can then be combined to form larger, more powerful environments that perform more complex tasks. From these, even larger and more sophisticated SE's can be built. SE-level abstractions for these complex environments can be based on the SE-level abstractions of the constituent SE's to provide higher-level functionality. In addition, this kind of aggregation will make implementation of larger SE's more feasible and manageable. For example, a smart building is constructed by integrating smart services provided by smart offices, smart classrooms, and smart meeting rooms within its physical confines. Similarly, a smart virtual office is built using more basic SE's such as smart virtual workspace, smart printer service, smart email service, smart directory service, and so forth, but is not necessarily confined to a single building. Our smart virtual department can be built using both smart buildings and smart virtual offices, among other services.

SE-level service abstraction and integration is achieved by three principal concepts:

- a *domain-specific, environment-wide shared state space* which defines the SE semantics, stores and manages SE states and events of common interest. This shared state space allows an SE to define semantically-rich user-service interactions. Based on the shared SE semantics, a smart service provisioned for the SE can specify how users and other services may interact with it (via its "service signature"). In addition, a user of the SE can also express its own preferences for interacting with the SE and its services ("user preferences") based on the shared semantics.



(a) name space hierarchy



(b) an example

Figure 1: SE Name Space.

- *execution context*, a mechanism that provides dynamic management of per-service-invocation state, notification of events defined on these states, and dynamic invocation support. Using the service signature and user preference, the SE can provide: customized service invocation and service behavior, semantically-rich service and user interactions via event filtering, event routing, and other dynamic execution context management mechanisms.

- a *flexible, declarative API* for user-service and service-service interactions that allows services to be invoked without *a priori* knowledge of their interface or specific details of the service itself. All interactions within the SE are based on this declarative API. Using the SE and its shared state space as the common broker, users and services of the SE can communicate and interact with each other via exchange of typed messages whose format is compliant with the shared SE semantics. The SE will mediate the interactions of the service and user by appropriately transforming these messages to meet the specific requirements of the service or user interface. Due to the declarative representation of shared SE semantics, users or services may change the specifics of their respective interfaces without breaking the SE. Furthermore, we can evolve an SE incrementally by extending its shared semantics without requiring changes in service or user interfaces.

Collectively, these concepts enable us to define and build SE's that allow seamless interactions customized by user preference, and provide services that are responsive to dynamic changes in the environment. Service integration enabled by the environment-wide shared state space and execution context mechanisms allow a greater degree of automation, convenience, efficiency, and personalization than is possible with a mere collection of individual loosely-coupled, communicating services. More importantly, through the SE-level service abstraction, we can potentially reduce the complexity of developing new smart services and environments, *and* alleviate the burden on users in managing service invocation and interactions. We believe that with structured SE service integration, larger and more complex SE's can be built in a scalable and extensible manner.

3.2 SE Core Architecture

In the previous subsection we have presented the key abstractions and concepts of the SE. In this subsection we describe the basic system constructs and architecture for building SE's. As mentioned earlier, an SE consists of a collection of smart services that perform domain-specific tasks. Each SE has a structured hierarchical name space that contains names and their associated attributes, e.g. a smart service contains names that refer to its interfaces, its attributes, etc. The names within the SE reflect the domain of that particular SE. For example, the smart virtual office SE contains specific names that refer to its services: smart printer, smart file system, etc. Such an SE could also define names for other common structures such as "documents" that may have SE-wide meaning.

The SE name space can be further extended by including names provided by individual services. Each service maintains its own service-specific name space (see Figure 1(a)) and will export names of common interest to the

SE. Names in the common name space have an agreed-upon SE-wide meaning or semantics. The shared semantics is represented by: the set of domain-specific SE system states, events defined on those states, and the collection of smart service "prototypes" that represent the services needed by the SE to perform its tasks. A prototype describes or summarizes what the services within an SE actually do at a high-level. The common SE name space and its contained service prototypes constitute the "signature" of the SE - it describes what the SE does, and how users of SE can interact with it. Likewise, each smart service of the SE also defines its own "service signature" using both the common SE name space and its own private service-specific name space: like the SE signature, the signature of a specific smart service describes how the SE and its users may interact with the service, including how the service should be invoked, names resolved, dynamic events handled, etc. Using the common SE name space, a user of the SE can also specify its own preferred "modes" of interaction with the SE and its services. This is referred to as the user preference (or user signature).

For example, a name in the common name space "bandwidth" can be expected to represent network bandwidth to all services and users within the SE. These shared names are useful in building other services or in supporting user preferences. For example, consider a smart virtual office SE. A smart file service may have many names in its own name space (e.g. its size, internal directory structure, etc.), but certain names will be of particular importance to other services or users (e.g. document types: gif, text, etc) and can be exported to the common name space (see Figure 1(b)). This would allow a smart email or smart printer service to determine whether a "document" was print-able or email-able. Similarly, users could request that email attachments of type gif be automatically filtered and removed.

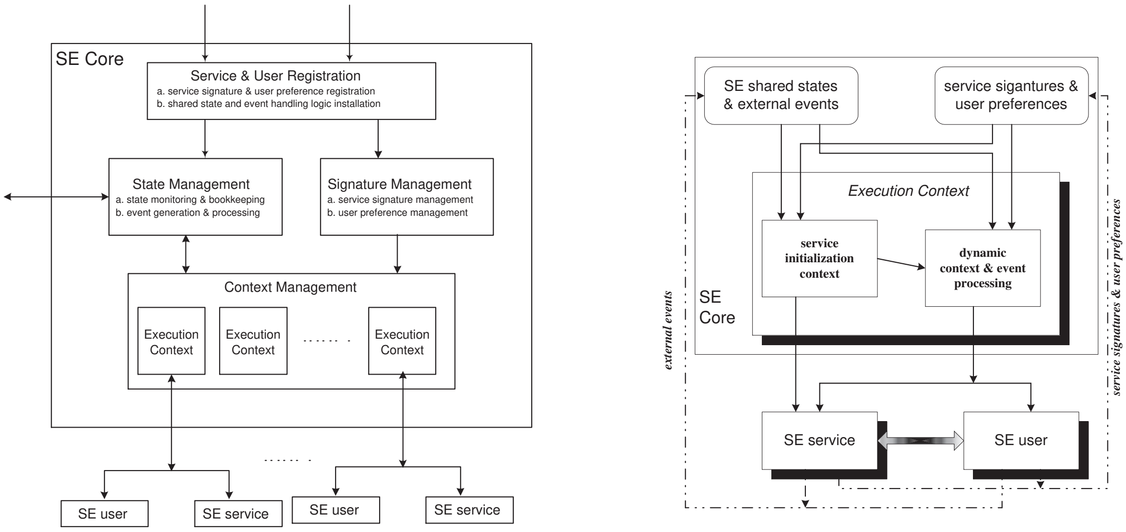
Figure 2(a) shows the basic system architecture of the SE core. The SE core architecture consists of four major components: SE service and user registration, SE core shared state and event management, SE signature management, and SE dynamic execution context management. The major functions of each component are described below. In the subsequent section, we describe possible implementation strategies, mechanisms, middleware, and important research problems we plan to investigate to establish the feasibility of the SE approach.

SE service and user registration: When a smart service is installed it must be first registered with the SE. The SE itself is registered at boot-time. To register, the smart service sends the SE its service signature. In the service signature, the service declares states and events defined in its own name space that may be of interest to other services or users of the SE (*exports*). It may export or provide values for names in the common name space or it can export names to extend the common name space. It also declares certain states and events defined in the SE common name space that are of interest to the service itself (*imports*). The service also specifies how it wants to interact with the SE and its users such as invocation modes and through dynamic event handling logic. This logic includes support for expressing interest in SE-wide events, and for detecting and generating its own service-specific events. Through the registration process, the SE "glues" together the service-specific name space and the SE common name space. The relevant information about the service obtained from the signature is passed to the SE shared state and event management and signature management components.

The user registration works in a similar fashion: when a user enters an SE, it may register to it by sending the SE its preference or signature (i.e., how a user prefers to interact with an SE and its services). From the user signature, the SE extracts relevant information such as states and events of interest to the user and its customized event handling logic. Because services may invoke other services within the SE, they too may specify a preference that reflects any customization required in the invocation of the other service. All information extracted during registration is stored in appropriate SE core components. The registration process is also a place where user-SE authentication can take place for environments which require secure access.

SE signature management: This component functions as a repository for storing service signatures and user preferences that have been registered with the SE. It is also responsible for updating the service signatures and user preferences whenever they are modified by smart services or users.

SE core shared state management: This component is responsible for monitoring and maintaining the shared state space of an SE. The shared state space consists of the states and events defined by the SE common name space as well as those exported by the SE smart services (see Figure 1). The SE monitors these shared states and generates



(a) core architecture

(b) dynamic context management

Figure 2: SE Core System Architecture.

appropriate events when changes in the shared states would affect the behaviors of other services or users. Events are *predicates* on shared states. SE-wide events may be triggered by simple or primitive predicates (e.g. shared state updates) or based on more complex predicates. These events are then passed to the SE execution context management for further processing. Event specification includes predicates and their firing action (whom to notify, and the type of event to generate). SE-wide events are defined as part of the SE-level service abstraction (in the SE signature) and are automatically delivered to the context management component. Individual service-specific events are generated and handled within the service execution context itself. The shared state is stored in SE *shared memory* which is an aggregate set of name value pairs associated with names in the common name space.

SE dynamic execution context management: When a user of an SE invokes a smart service, the SE creates a dynamic execution context for this user and the smart service. The execution context is initialized based on the service signature and user preference, where a "customized view" of the SE shared states is created for this user-service interaction (see Figure 2(b)). Customized event processing such as event registration, detection, filtering, routing, and handling are also created using information extracted from the service signature and user preference. During service execution, a change in the shared states of the SE triggers an SE-wide event that may affect the behavior of either the user or smart service or the way they interact. In either case, these SE-wide events will be passed to the SE dynamic execution context management component which performs routing and filtering. If the event is of interest to a particular dynamic execution context, then it will be automatically routed to it. Otherwise, it will be filtered. Events sent to an individual context, may trigger other more specific events.

4 Realizing the SE: Implementation, Middleware, and Research Challenges

4.1 Naming and Shared Semantics

- *How is the SE name space represented to allow integration and extensibility?*

The SE name spaces are hierarchical in nature. We propose the use of self-describing XML DTD schema for the representation of names, and the structured hierarchical name space within the SE. XML does not impose any pre-defined semantics, is an open standard, supports extensibility, and allows for flexible interactions. The absence

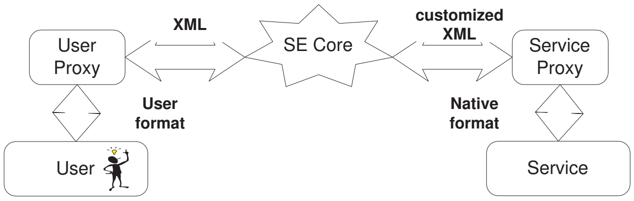


Figure 3: XML Transformation Using Proxies.

of pre-defined semantics allows the SE and contained services to interpret the meaning of the names in a domain-specific, environment-wide manner. Extensibility is crucial because the name space may change dynamically, e.g. as new services get registered or re-registered, or modified. XML will also be used to define interactions between all services in an SE (between SE core and non-core services and users via their proxy). This allows services to interact without explicit knowledge of the interface or low-level details of the service. XML "objects" are transformed into typed serialized messages. Further transformation from XML to native protocol formats expected within the service implementation can be done via proxies (Figure 3).

XML also allows the definition of a second hierarchy based on DTD types. For example, we can define a DTD schema for a "generic" smart service which contains attributes "inherited" by any smart service, e.g. a smart printer service. All entities within the SE core including events, generic smart services, service invocations, etc. will be typed by XML DTD schema. Research is needed to determine the structure and contents of these generic XML DTD's within the SE, and the mandatory contents of a basic generic SE name space.

• *How are names resolved?*

Research is needed to develop naming and directory service middleware that can scale for large SE's containing thousands of names and possibly millions of attributes.

4.2 Shared State and Event Management

• *How to efficiently and scalably represent the shared memory of millions of entities that can be shared across many contexts?*

Data structures that efficiently support large shared memories containing attributes, user and service signatures will be needed. Such data structures must also support the division of shared memory into overlapping contexts.

• *How to efficiently detect and generate SE-level events?*

SE-level events must be generated when predicates defined on shared-memory fire them. Value updates are the most primitive (and frequent) kind of event that must be efficiently managed. Since CPU resources within the SE core are precious, efficient event execution middleware is needed. Compact event representation in terms of XML DTD's must also be developed.

4.3 User Preference and Service Signature Registration

• *What is contained in a user preference or signature?*

User preference must contain sufficient information to customize the user's interaction with services within the SE including user choices, decisions, and priorities. The signature is divided into two parts: service-specific preferences and SE-wide preferences. Service-specific preferences describe the user-preferred input sources and output sources for delivering arguments to an invocation and for returning outputs respectively (for each service of interest). This enables workflow customization by the user. In addition, personalized information used to select and customize the invocation is also specified. SE-wide preferences refer to more general preferences such as SE-level event notification (i.e. the user wishes to receive particular SE events), attributes the user wishes to import from the

SE as part of its context, and attributes the user wishes to export to the SE to augment the common name space. Research is needed to determine the minimal set of mandatory fields required for user preferences.

• *What does the service signature API of a smart service contain?*

The service signature contains the following information: *service description*, *service invocation*, *service resolution*, *service context*, and *service events*. The service description contains an unrestricted list of attribute value pairs which describe properties of the service. Service invocation describes the input/output relationships of the service. Inputs to the service describe the set of arguments to an invocation. Similarly, outputs from a service are the list of recipients to receive the output. Service resolution specifies which lookup and resolver services are to be used to locate and resolve instances of this service. Service context has two parts: imported and exported context. Imported context consists of the set of named attributes of interest to this service. Exported context are the set of named attributes that will be exported to the common-name space. Exported context specification must include a service that will be used to produce values for the attribute (perhaps this service). Service events specify the SE-wide events of interest to this service, and allow more specific events to be detected and generated. Research is needed to determine the minimal set of mandatory fields required for a smart service signature.

• *What does the service signature API of the SE itself contain?*

The SE presents a high-level service abstraction that is summarized by its own service signature. The SE signature specifies its service description, service invocation, service resolution, service context, and service events, all SE-wide. The service description contains environment-wide attributes that describe the types of services and shared information this SE will contain. Service invocation describes the various operations available to services and users of an SE, and how they can be invoked. For example, the SE middleware handles service requests, signature registration, allows users and services to dynamically change their signatures, etc. Service resolution refers to the default resolution service. The common name space is initially constructed from the set of exported names in the SE service context. An SE may also import names from other SE's and this would be specified as imported context (see the final subsection). This is how an SE serves as a building block to construct larger more functional SE's. Service events (SE-wide events) are also specified within the SE signature. Research is needed to determine the minimal set of mandatory fields and mandatory services required within an SE signature.

• *How to compactly represent preference and service signature objects for efficient parsing, attribute and event extraction, transmission, and storage in the SE core?*

Every entity in the SE is represented by an XML DTD. Compact representations that can be communicated efficiently, but also support rapid parsing and processing, are needed.

4.4 Dynamic Execution Context

• *How to design a scalable event detection and propagation engine?*

The dynamic execution context management component receives SE-wide events and must perform routing and filtering to determine which individual contexts need to receive the event. Because this component must interface to a potentially large number of individual contexts, efficiency is paramount. The individual contexts also must handle these events and possibly generate their own specialized events in response to SE-wide events.

• *How to efficiently represent the service context?*

To perform routing and filtering of events and to allow a service context to receive specific state information on the attributes it uses, the representation of the service context is key. Service contexts will overlap each other when they contain attributes of common interest. The context management component must maintain the contents of each service context in an efficient and compact manner.

• *How to manage the customized invocation?*

Invocation requests must be combined with the service signature and user preference to generate a specific context for this invocation and to invoke the service on behalf of the user. This includes routing inputs to the service instance and outputs from the service instance. Inputs may come from sources specified by the user, and outputs may be sent to one more services as also specified by the user, allowing for workflow customization. XML objects will used to represent user requests, preferences, service signatures, invocations, inputs, and outputs. Middleware to support invocation management within the service context is needed.

In addition to the research challenges, there are many other issues such as SE management, service discovery, access control and security, and user privacy that need to be addressed. Due to space limitation, we will not touch on these issues.

5 Related Work and Conclusions

Recent years have seen a great flurry of research activity in pervasive computing and related areas from both academia and industry. Taking advantage of various emerging wireless communications technologies [18, 7, 31, 36, 14, 27], Sun Microsystem developed the Jini service [4] that enables devices to discover and communicate with each other using Java [23]. Based on the Jini substrate, a storage service - JavaSpaces [21] - has also been developed to provide a storage facility for *persistent* Java objects. Similarly, IBM developed T-space [47] which provides a global uniform communication and persistent storage infrastructure for "strongly-typed" objects (tuples). Along a different axis, many research and development efforts have been devoted to integrate Web and other computing services (see, e.g., [12, 17, 32, 37, 42, 46]), in particular, using XML-based technologies [9]. Research in novel naming and directory services has also been conducted, see, e.g., [1, 15]. Security is also a key issue in this environment (see, e.g., [6, 33, 38, 40]). Another active area of research in pervasive computing is human-computer interface, where various techniques for providing easy-to-use, dynamic, cognitive, and intelligent human computer interaction are being developed (see, e.g., [2, 3, 8, 19, 20, 29, 30, 35, 39, 45]). Various issues in personalization have been also studied [13].

In addition to addressing various aspects of pervasive computing such as communications, persistent storage, human-computer interface, new computation models and system architectures for mobile and pervasive computing have also been proposed (see, e.g., [5, 10, 11, 16, 22, 24, 25, 26, 41, 44]). Perhaps the most closely related to our effort is the `one.world` project at the University of Washington [34, 24]. In this project, a system architecture for pervasive computing is outlined based on three principle concepts: tasks, tuples and environments. In contrast, our model adopts a service-oriented, domain-specific approach: we advocate *smart services* and *smart environments* (integrated collections of services) as building blocks for developing a scalable middleware infrastructure for seamless network computing. Toward this end, we advance a common set of fundamental abstractions and a generic SE core architecture for service integration and user customization, with emphasis on ease of service construction, ease of service deployment, and ease of user personalization. We see much of the related work as complementary and plan to adopt and leverage techniques and solutions where appropriate, especially open standards, developed by the community at large.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. "The Design and Implementation of an Intentional Naming System." In *Proceedings of the Seventeenth Symposium on Operating Systems Principles*, pp.186-201, Kiawah Island Resort, South Carolina, Dec. 1999.

[2] G. Abowd, C. Atkeson, I. Essa, and K. Ramachandran. "Buidling a Large-Scale Ubiquitous Computing Interface." 1998. A Proposal Submitted to the National Science Foundation Major Research Instrumentation Program, `www.cc.gatech.edu/fce/pubs/BaLSUCI.pdf`.

[3] G. Abowd. "Ubiquitous Computing: Research Themes and Open Issues from an Appications Perspective." Technical Report GIT-GVU 96-24, GVU Center, Georgia Institute of Technology, Oct. 1996. Available from `www.cc.gatech.edu/fce/publications.html`.

[4] K. Arnold, B. O'Sullivan, R.W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. Addison-Wesley, 1999.

[5] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. "Challenges: An Application Model for Pervasive Computing." In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000).*

[6] E. Belani *et al.* "The CRISIS Wide Area Security Architecture." In *Proceedings of USENIX Security Symposium,* 1998.

[7] Bluetooth SIG. *Specification of the Bluetooth System*. Dec 1999, `http://www.bluetooth.com`.

[8] M. Brand and I. Essa. "Causal Analysis for Visual Gesture Understanding." In *Proceedings of AAAI Fall Symposium on Computational Models for Integrating Languages and Vision,* 1995.

[9] T. Bray, J. Paoli, C.M. Sperberg-McQueen. *Extensible Markup Language (XML)*. W3C Proposed Recommendation, Dec. 1997. `www.w3.org/TR/PR-xml-971208.`

[10] L. Cardelle. "Abstractions for Mobile Computations." In J. Vtek and C.D. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*. Vol. 1603 of *Lecture Notes in Computer Science*, pp.51-94, Springer-Verlag, 1999.

[11] L. Cardelli and A. D. Gordon. "Mobile Ambients." In M. Nivat, editor, *Foundations of Software Science and Computational Structures.* Vol. 1378 of *Lecture Notes in Computer Science*, pp.140-155, Springer-Verlag, 1998.

[12] Classroom 2000. `www.cc.gatech.edu/fce/c2000`.

[13] *Commmunications of ACM.* "Special Issue on Personalization." Vol. 43, No. 8, Aug. 2000.

[14] L.M. Correia, R. Prasad. "An Overview of Wireless Broadband Communications." *IEEE Communications Magazine*, pp.28-33, Jan. 1997.

[15] S.E. Czerwinski, B.Y. Zhao, T.D. Hodes, A.D. Joseph, and R. H. Katz. "An Architecture for a Secure Service Discovery Service." In *Proceedings of the Fifth ACM/IEEE Internation Conference on Mobile Computing and Networking*, pp.24-35, Seattle, Washington, Aug. 1999.

[16] A. Dearle. "Towards Ubiquitous Environments for Mobile Users." *IEEE Internet Computing*, pp.22-32, Jan.-Feb. 1998.

[17] A. Dey, G. Abowd, and A. Wood. "Cyberdesk: A Framework for Providing Self-Integrating Context-Aware Services." In *Proceedings of the 1998 Intelligent User Interfaces Conference (IUI'98)*, pp.47-54, 1998.

[18] IEEE 802.11. "Wireless Access Method and Physical Layer Specification." New York, Sep. 1994.

[19] I. Essa, S. Basu, T. Darrell, and A. Pentland. "Modeling, Tracking and Interactive Animation of Faces and Heads Using Input from Video." In *Proceedings of Computer Animation Conference 1996*, pp.68-79. Jun. 1996.

[20] I. Essa and A. Pentland. "Coding, Analysis, Interpretation, and Recognition of Facial Expressions." *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 19(7):757-763, 1997.

[21] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice.* Addison-Wesley, 1999.

[22] D. Garlan. "Architecture for Pervasive Computing." In *Proceedings of the Workshop on Architectures for Ubiquitous Computing.* Limerick, Ireland, June, 2000.

[23] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification.* Addison-Wesley, 1996.

[24] R. Grimm, T. Anderson, B. Bershad, and D. Wetherall. "A System Architecture for Pervasive Computing." In *Proceedings of the 9th ACM SIGOPS European Workshop*, pp.177-182, Kolding, Denmark, Sep. 2000.

[25] R. Grimm *et al.* "System Directions for Pervasive Computing." In *Proceedings of Hot Topics in Operating Systems (HotOS-VIII)*, Elmau, Germany, May 2001.

[26] O. Gruber and R. Konuru. "An Application Model and Environment for Personal Information Appliances."

[27] W. Honcharenko, J.P. Kruys, D. Y. lee, and N.J. Shah. "Broadband Wireless Access." *IEEE Communicatios Magazine*, pp.20-26, Jan. 1997.

[28] M. Esler, J. Hightower, T. Anderson, and G. Boriello. "Next Century Challenges: Data-Centric Networking for Invisible Computing." In *Proceedings of the Fifth ACM/IEEE Internation Conference on Mobile Computing and Networking*, pp.256-262, Seattle, Washington, Aug. 1999.

[29] J. Mankoff and G. D. Abowd. “Domisilica: Providing Ubiquitous Access to the Home.” Technical Report GIT-GVU-97-17, GVU Center, Georgia Institute of Technology, Sep. 1997. Available from www.cc.gatech.edu/fce/publications.html.

[30] J. Mankoff and G. D. Abowd. “Bringing People and Places Together.” In *Proceedings of the 1998 Spring AAAI Symposium on Intelligent Environments*, 1998.

[31] R. Mettala. *Bluetooth Protocol Architecture, Version 1.0.* Bluetooth Special Interest Group, Aug. 199. www.bluetooth.com/v2/document.

[32] Microsoft .Net Homepage, http://www.microsoft.com/net/default.asp.

[33] B.C. Neuman. “Proxy-Based Authorization and Accounting for Distributed Systems.” *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[34] one.world, http://one.cs.washington.edu.

[35] A. Pentland. “Smart Room.” *Scientific American*, 274(4):68-76, April 1996.

[36] C. Perkins, et al. “IP Mobility Support.” *IETF RFC 2002*, Oct. 1996.

[37] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. “Virtual Network Computing.” *IEEE Internet Computing*, pp.33-38, Jan.-Feb., 1998.

[38] R. L. Rivest and B. Lampson, “SDSI-A Simple Distributed Security Infrastructure.” http:theory.lcs.mit.edu/cis/sdsi.html, 1996.

[39] A. Smailagic and D.P. Siewioerk. “User-Centered Interdisciplinary Design of Wearable Computers.” *ACM Mobile Computing and Communications Review*, 3(3), 1999.

[40] J. G. Steiner, et. al, “Kerberos: an authentication service for open network systems.” In *Proceedings of USENIX,* 1988.

[41] A. Tripathi, T. Ahmed, and N. Karnik. “Experiences and Future Challenges in Mobile Agent Programming.” Submitted to *Microsystems*, Oct. 2000.

[42] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. “WebOs: Operating System Services for Wide Area Applications.” In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing,* pp.52-63, Chicago, Illinois, July 1998.

[43] S. Verdu. “Wireless Bandwidth in the Making.” *IEEE Communications Magazine*, pp.53-58, Jul. 2000.

[44] Z. Wang and D. Garlan. “Task-Driven Computing.” Technical Report CMU-CS-00-154, Carnegie Mellow University, School of Computer Science, May 2000.

[45] M. Weiser. “The Computer for the 21st Century.” *Scientific American*, 265, 3 (September 1991), pp.94-104.

[46] A. Wood, A. Dey, and G. Abowd. “Cyberdesk: Automated Integration of Desktop and Network Services.” In *Proceedings of the 1997 Conference on Human Factors in Computing Systems (CHI’97)*, pp.552-553, 1997.

[47] P. Wyckoff, S.W. McLaughry, T.J. Lehman, and D.A. Ford. “T Spaces.” *IBM System Journal*, 37(3):454-474, 1998.