Practical Web-Based Smart Spaces

Researchers from Nokia propose a Web-based framework that applies a resource-based HTTP style called Representational State Transfer (REST) to enable smart spaces to support pervasive applications in various devices.

Christian Prehofer, Jilles van Gurp, Vlad Stirbu, Sailesh Sathish, Pasi P. Liimatainen, Cristiano di Flora, and Sasu Tarkoma
Nokia

Mobile devices are evolving into hubs of content and context information. Therefore, our work focuses on pervasive applications in smart spaces that use locally available connectivity and device discovery. This approach allows, for example, sharing content and offering services locally with direct connections between devices. Although many research projects have shown the potential of such applications, the results of this research haven't yet become widely deployed. An important reason for this is interoperability—a fundamental requirement for practical smart-space applications. Demonstrating smart applications in a research lab environment is straightforward with a few limited devices but is challenging when there are hundreds of different devices with many different operating systems and runtime environments. This is a well-known challenge for ubiquitous and pervasive computing; a recent survey features 29 different pervasive computing platforms up to 2004.[1]

Our approach is to address practical smart-space deployment using Web technology, which is becoming an integral part of mobile devices as consumers demand increased Web access. In this article, we outline solutions for several challenges in smart-space applications. First, Web technologies must be integrated with local smart spaces and made available on more mobile devices. Although many approaches have used Internet technology, we use Web services based on Web 2.0 HTTP and an HTTP style called Representational State Transfer (REST) as middleware to mashup services, both between devices and within a device. Second, resources in a smart space must be able to find one another. A heterogeneous environment makes this resource discovery, and subsequent communication between resources, challenging. Third, privacy and security are major challenges for services and application mashups in smart spaces. Currently, application mashups are often poorly secured.[2] Mobile devices typically involve sensitive information such as location and other personal content. Devices in the smart space engage in many complex interactions with services on other devices and on the Internet that must be properly secured. Widely used Internet practices, which often require manual entry of passwords and authorization of services, aren't adequate here. Fourth, the Web browser model of consuming services doesn't allow access to local context and content on the device. Fortunately, several upcoming standards will permit such context access from the Web browser.

We provide solutions to meet these challenges through a REST-based framework that includes support for finding relevant local resources, authorizing access to resources with group-based security, and sharing context information on the device. To demonstrate the feasibility of providing pervasive smart-space services, we've implemented a prototype smart-space shopping mall

Published by the IEEE CS ■ 1536-1268/10/$26.00 © 2010 IEEE

Related Work in Smart Spaces

Earlier approaches such as CoolTown have applied basic Web architectures to ubiquitous applications for PDAs.[1] We now see the opportunity to use the traditional Web as well as Web 2.0 technologies as a platform for providing smart-space services. Various researchers have recently proposed the Representational State Transfer (REST) HTTP style for pervasive applications because of its simplicity and interoperability.[2,3] Proposed enablers in this environment include the Semantic Web, ontologies and SOAP (Simple Object Access Protocol)-based Web services, and agent systems.[4,5]

Our approach advocates lightweight solutions that use established standards. We don't use SOAP-based Web services because of their complexity and lack of suitability for mashup applications. Many consumer Internet services omit SOAP support for this reason as well.

REFERENCES

1. P. Debaty and D. Caswell, "Uniform Web Presence Architecture for People, Places, and Things," *IEEE Personal Comm.*, vol. 8, no. 4, 2001, pp. 46–51.
2. D. Coffman et al., *Modeling and Managing Pervasive Computing Spaces Using RESTful Data Services*, tech. report RC24344, IBM Research, 2007.
3. W. Drytkiewicz et al., "pREST: A REST-Based Protocol for Pervasive Systems," *Proc. IEEE Int'l Conf. Mobile Ad-Hoc and Sensor Systems* (MASS 04), IEEE Press, 2004, pp. 340–348.
4. C. Endres, A. Butz, A. MacWilliams, "A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing," *Mobile Information Systems*, vol. 1, no. 1, 2005, pp. 41–80.
5. X. Wang et al., "Semantic Space: An Infrastructure for Smart Spaces," *IEEE Pervasive Computing*, vol. 3, no. 3, 2004, pp. 32–39.

setup that integrates these solutions. (The "Related Work in Smart Spaces" sidebar discusses other approaches to smart spaces.)

Advantages of Using Web Technology

In our proposed approach, devices offer and consume services using Web technology. The Web has already proven to be a highly interoperable software platform. Although not every device can offer services, they'll be able to consume services and provide content and context information for those services. Web technology is an essential enabler for combining locally provided services with other Internet services. For instance, users could share geotagged content locally and load maps and other content from third-party Internet services. We can extend the current success of application mashup technologies in Internet applications to pervasive services in smart spaces. Currently, application mashup typically uses HTTP in REST style, combined with RSS or Atom feeds.

Another important aspect is that the necessary skills to create Web-based applications are now widely developed, and an abundance of tools and developer and vendor support are available for Web-based development. Technologies such as Web servers and content management systems are becoming available on mobile devices as well. In addition, solutions for hosting Web-based services on mobile devices (such as the Nokia Mobile Web server, mikz.com, and httpd4mobile) have recently emerged. Moreover, the popularity of push email and chat on mobile devices shows the feasibility of semi–real-time delivery of messages to devices. However, fully integrated and optimized solutions for mobile devices are still under development. In addition, mobile devices have limitations, such as more-limited memory, energy, and wireless-transmission resources. Mobile devices are used in a personal context and have several additional requirements, which differ from those of PC-centric devices.

Web and Resource-Based Smart Spaces

A smart space is a heterogeneous, local environment comprising many different devices with varying capabilities (see Figure 1). Being Web-based, a smart-space network consists of devices capable of accessing, and optionally providing, HTTP-based Web services. In practice, this includes most devices with a network connection (Wi-Fi or telephone network) and TCP/IP support. For low-end devices without this capability, we use a proxy solution in which high-end devices (such as phones or access points) proxy functionality of more-limited devices (sensors, Bluetooth peripherals, and so on). With this approach, we can target a large portion of the current mobile and portable-devices market.

We distinguish between devices that host Web-based services and clients that only consume services. Clients are either Web browsers accessing an HTML- or XHTML-based Web application in the smart space or on-device applications that access smart-space services through their APIs. The key advantage of using HTML-based user interfaces (UIs) is that they can work well across devices with a sufficiently capable Web browser. In addition, deployment is easier because such UIs don't require software installation. A native application might provide better usability, but at the cost of sacrificing portability, and would require installing the software before it could be used.

Both Web services and Web applications depend on Web servers. The

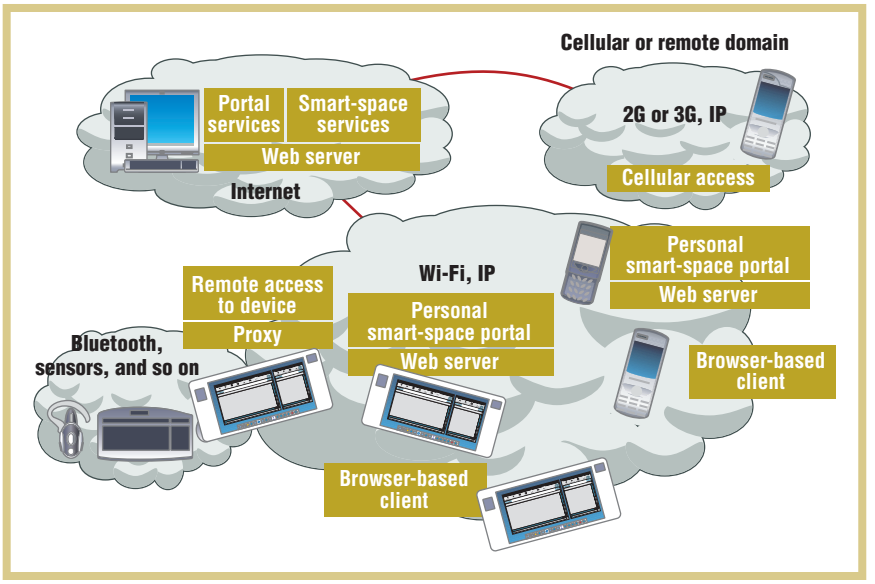


Figure 1. Smart-space network architecture. In this architecture, mobile devices can host smart-space portals in different network domains, connected by IP technology. Any device with a browser can access these portals.

distinction is that a Web service provides an API, whereas a Web application provides a browsable application UI (usually HTML based). Servers can be embedded in, for example, a mobile phone or hosted on the Internet. One advantage of running an embedded Web server in a mobile device is that such a server can access on-device functionality and resources. For example, the Nokia Web Server and httpd4mobile come with a portal that can access camera and application data such as contacts and calendars. Users can reach this Web server, even if firewalls and network address translators (NATs) are present (for instance, in mobile operator networks), by employing a permanent connection to a proxy server, which passes any incoming traffic to the device over the open connection. This is conceptually similar to how JavaScript techniques such as Ajax can turn a passive Web application in the browser into a message-processing server. The proxy also provides a URL for the device.

Figure 2 shows our software architecture for a mobile device hosting a Web portal, including the Web platform and middleware services. Distributed operation is realized by querying a search engine for services and accessing those services.

Smart-space services in our architecture provide REST APIs. Thus, entities such as people, content, devices, and services are all resources with a uniform resource identifier (URI), which can be manipulated using simple operations provided by HTTP such as **get**, **post**, **put**, and **delete**. The main advantages of using REST APIs are that they're suitably lightweight for implementation on limited devices and they align well with existing APIs. Because our approach is based on the Web and the REST style, we've reused existing Web components (such as feed aggregators, content management software, Web application frameworks, and databases) in our shopping mall demo. Another key benefit of REST-style Web services is that our target market of mobile and portable devices can easily support them.

Locating Resources

Smart-space services are similar to location-based services on the Internet in that they're typically scoped to a particular physical space such as a room, a moving vehicle, or a building. However, they differ in that such spaces aren't easily identifiable via simple attributes such as GPS coordinates.

In our setting, people, devices, and resources can come and go, and they need to constantly adjust to the dynamic nature of the smart space. Locating entities requires indoor maps, points of interest, and an indoor-positioning infrastructure. To foster and support easy mashup of indoor location-based services, we've adopted solutions that are simple to integrate with existing Web technologies and applications, such as making services and their geospatial data available through REST APIs and widely used formats such as Atom and GeoRSS feeds. We developed our system by integrating commodity open source geographic information system (GIS) software and novel research prototypes developed in our labs, including an indoor-positioning system based on Wi-Fi.[3]

As is common for indoor-positioning systems, our positioning system represents locations using spatial relations between buildings, floors, sections, and rooms. We represent these locations using a URL schema. For example, http://nokia.com/locations/helsinki/ruoholahti/4/A/401 references room 401 in one of the Nokia premises. (Note that the location system is not currently running on a public domain.)

Referencing resources using a URL is a concept that's native to the Web and is an important aspect of the REST architectural style. It's also an important concept in Web 2.0, where content is often tagged with keywords that are typically part of a namespace that a URL represents. The URL with /<keyword> appended then serves as a tag URL to refer to the tag (for example, http://delicious.com/tags/smartspace). Resources can be indoor-geotagged (which is similar to geotagging information with GPS coordinates) with location URLs like

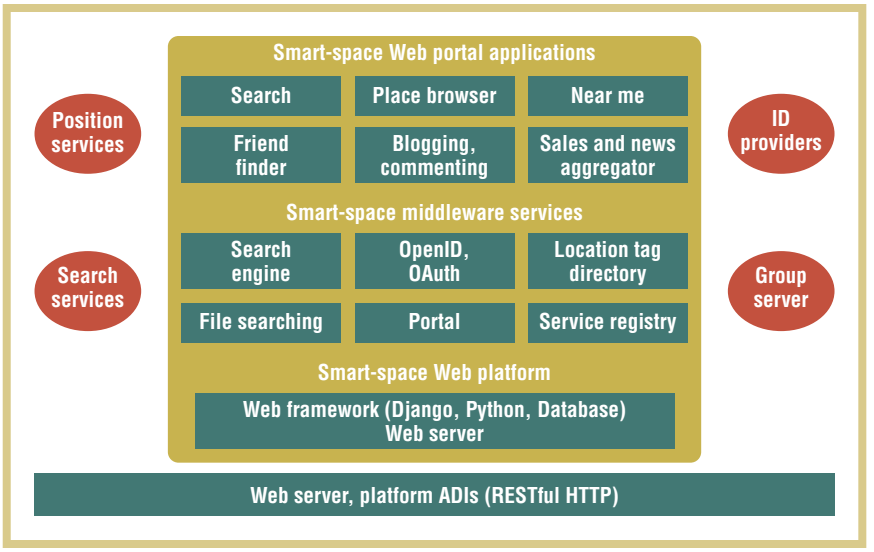


Figure 2. Software architecture overview. Our architecture is implemented using a variety of commonly used technologies intended for Web development for which we managed to find usable ports on the Nokia N800 devices used in our system.

the keyword-based URLs used by Delicious.com. For example, we indoor-geotag items in an Atom feed by simply including link elements with the rel="tag" attribute set.

The tag URLs refer to resources in a location tag directory that provides meta-information about the locations, such as GPS coordinates, name, and description. Services can manipulate this directory using the Internet Engineering Task Force's Atom Publishing Protocol (RFC 5023). This is a REST-based protocol for performing create, read, update, and delete operations on resources that is complementary to the Internet Engineering Task Force's RFC for Atom feeds (RFC 4287).

Finding Resources

A key issue for interacting with resources in a smart space is finding out their URLs. Considerable research has focused on service registry and discovery. Although researchers have proposed various solutions, the Web and its services function mostly without such solutions—aside from the Domain Name System (DNS), which is for looking up domain names. A smart space is far more dynamic than the current Web, and a service registry or discovery mechanism is necessary to create mashups that use locally available resources.

In our system, we use a combination of both a service registry and a discovery mechanism, and we decouple the problem of finding resources from that of registering or discovering them. To find resources in the smart space, we use a search engine based on Apache Lucene (http://lucene.apache.org) that indexes resources, including their tags, which encapsulate information about their type, location, ownership, and so on. Discovering resources then becomes a matter of specifying the right tags in a search query and using the resulting resource URLs. Services can access the search service through the Internet or discover it in a local network via Multicast DNS (MDNS).

The search engine can be populated with resources in various ways: it can crawl networks for resources; use traditional discovery mechanisms, such as universal plug and play (UPnP) or MDNS, to discover resources in local networks; or be used as a traditional service registry (local or Internet based). In our prototype system, we use a combination of these approaches. For example, we use discovery to add UPnP media-server-advertised content to the search service. Other services register directly with the search service, and resource consumers look up these services through simple queries to the search service.

Combined with the location tag directory, this solution provides a flexible system for tagging resources such as content objects, service end points, and other information with location URLs, and for retrieving references to resources by querying the search engine. We also use tagging for associating other semantics with resources. We use service-type tags, content-type tags, group membership tags, and so on. Consequently, this tagging technique is also suitable for nonlocalized, virtual smart spaces (such as clusters of devices and services belonging to a person, company, or group of friends). Location is just one of many possible search criteria.

Protecting Resources

Ubiquitous context information, based on location and sensing of the surrounding environment, is typically privacy sensitive. Similarly, the ability to act on the physical world and the entities in it must be restricted to prevent abuse by unauthorized users or services. Consequently, access to resources should require authorization on the basis of some specific criteria. There is considerable work on security and access control for pervasive systems and discovery protocols.[4] Our goal is to use existing, Web-based application mashup techniques, which also combine easily with existing Internet technologies and services. However, current mashups are not very secure and don't sufficiently support our setting with many distributed resources.[2]

Resources are accessed by different services on behalf of users. These include resources owned by the user,

those owned by other users, and those provided by various services. A typical example is giving access to your current location, which some location services provide, to other services. This becomes more complex for a friend-finder application that must access your friends' locations on different location services. The application must authenticate with each location service, which in turn must check authorization rules to make sure you're allowed to see your friend's location.

We needed a solution for this that's lightweight and simple and that doesn't rely on a centrally owned and provisioned identity service. In addition, it must integrate well with our resource-based architecture, scale to large groups of resources and loose associations, and be easy to support across a wide range of existing devices.

There is considerable work in this area of security, including, for example, the Liberty Alliance Identity Federation Framework and several trust management systems such as SPKI/SDSI (Simple Public Key Infrastructure, Simple Distributed Security Infrastructure), which are distributed in that they permit separate, local namespaces. Access control is possible over several kinds of relations such as "children of Dad's friends," where the local names are maintained locally. However, these solutions don't fully meet our requirements of being lightweight and mashup friendly.

Two recently emerged Web-based protocols, OpenID and OAuth,[5,6] rely on a novel, more flexible, decentralized approach to authentication and authorization and thus can provide the foundation for a solution that meets our requirements.

OpenID is a lightweight, single-sign-on, identity federation protocol that relying parties (that is, services authenticating users via OpenID) can use to authenticate users with an identity provider. To sign in, the relying party verifies whether the user's client (typically, a browser) has authenticated the user with the identity provider that the user specifies. An important aspect of OpenID is that it employs identity URLs to identify users.

OAuth is a similar protocol for allowing services to authenticate one another on behalf of users. The result of a successful authentication with OAuth is a token that a service can use to access another service. OAuth is especially popular for implementing mashup applications involving services from different service providers.

Another important feature of OpenID and OAuth is that they rely on URLs to specify trusted entities. Hence, they fit nicely with our REST-based architecture. We extend these protocols to cover scenarios with large amounts of resources, people, and services. To do so, we organize resources in groups, which themselves are resources.

The application must authenticate with each location service, which in turn must check authorization rules to make sure you're allowed to see your friend's location.

A group server is technically similar to the location tag directory discussed earlier, and a group is simply a list of URLs; each group has a public key and a private key. The central idea is to authorize access to resources on the basis of group membership. The group server can issue a signed proof of group membership. This membership assertion is then included in requests to resources that can simply verify the signature.

We integrate the group server mechanism with the location-tagging mechanism and search architecture discussed earlier, to implement a system in which services can employ groups and membership assertions to find and access the resources they're authorized to use. A modified OpenID provider keeps track of a user's group memberships, and an OAuth-like mechanism between services passes on membership proofs. Group members can be people represented by their OpenID URLs, services represented by their service end-point URLs, or any other type of resource identified by a URL.

Figure 3 illustrates our approach with a friend-finder example. We assume that the friend finder securely obtains membership tokens in advance. A nonce prevents replay attacks. Any public keys fetched can, of course, be cached. In addition, HTTPS certificates can verify ownership of public keys being exchanged.

A key benefit of this approach is that it can vastly reduce the number of HTTP redirects in the browser that would otherwise complicate use of either OpenID or OAuth with more than a handful of resources. With OAuth, the browser would have to redirect to each resource individually, just to verify authentication. Therefore, using verifiable and cacheable assertions considerably reduces network overhead for clients, which is essential in a smart space.

The approach is decentralized because it decouples service providers from identity providers and group servers, which can thus all be flexibly reused with one another in the smart space. For example, a location service can discover the user's list of friends and their preferred identity provider and enforce a rule that the user's friends may access their location when another service is doing so on their behalf. Furthermore, it's easy to adopt

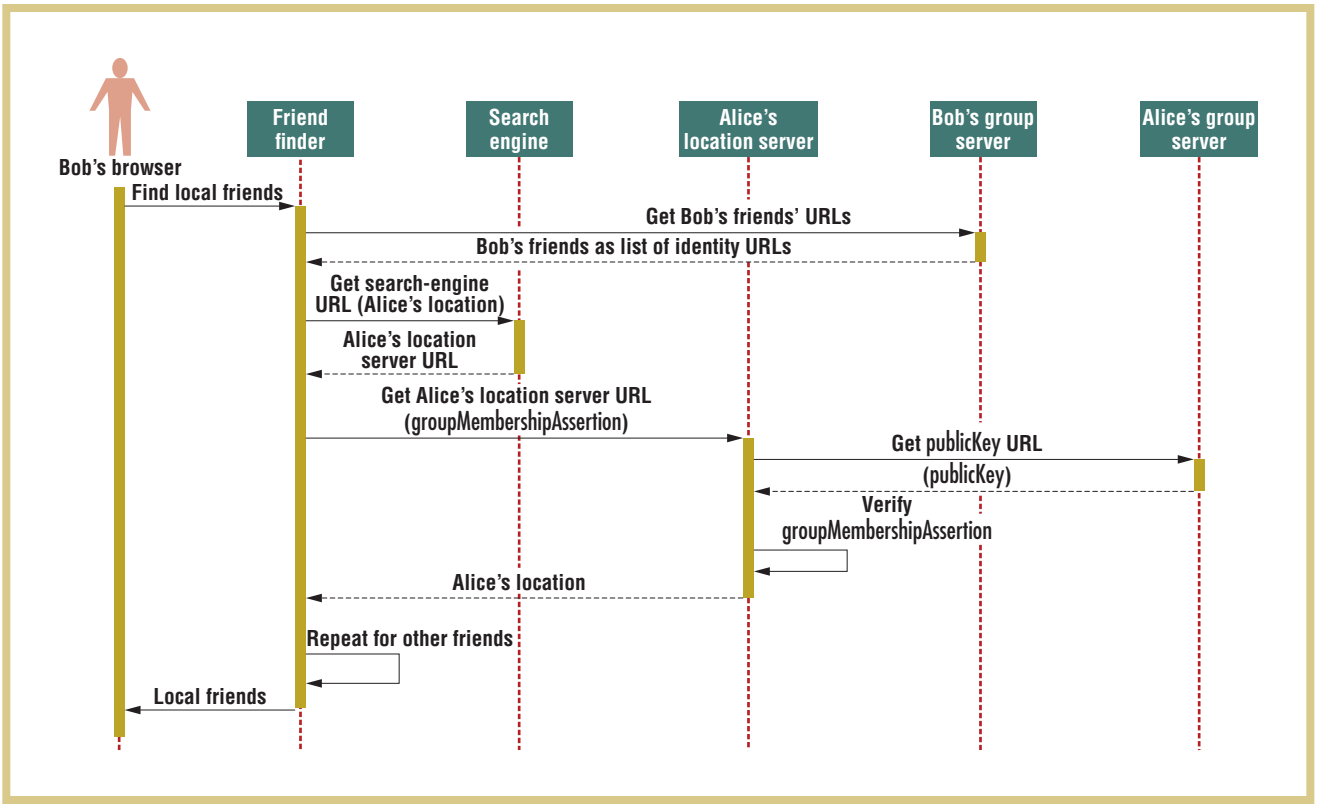


Figure 3. Sample interaction for a friend finder with multiple location and group servers. The friend finder uses Alice's identity URL to query the search engine for her location service provider. Then it uses the membership token that proves that Bob is a friend of Alice to authenticate for the location service, which returns Alice's location after verifying the signature.

this protocol for service providers because it closely resembles the way OAuth and OpenID work. Finally, this protocol reduces the number of user interactions needed (for example, permissions and confirmations) by relying on assertions instead.

This approach achieves appropriate levels of security and trust similar to other public- and private-key-based security mechanisms that the ubiquitous-computing research community has proposed. Our system decentralizes trust and relies heavily on the choices of, for example, identity providers and users regarding which groups to trust, which services to trust, and so forth. A smart space is too dynamic and heterogeneous to orchestrate all this centrally. So, decentralization is a suitable way to help smart-space participants self-organize along alliances, relations between friends, and so on.

Context Information for Browser-Based Applications

Although not all devices will be able to provide services, most will be able to consume services. Adaptive Web applications used on such devices need access to device and user context information available on the user's device. To provide support for this local data sharing, we've experimented with Delivery Context: Client Interfaces (DCCI) from the World Wide Web Consortium (W3C).[7] DCCI is based on W3C's Document Object Model (DOM), which is an API with an underlying tree representation for accessing and manipulating HTML and XML documents. DCCI extends the standard DOM tree and API to provide a context tree in which each context source has its own node representation. A directed-event-propagation model based on DOM events allows remote capture of specific events and change notifications from other devices. In addition, DCCI supports static and dynamic properties (local or remote). DCCI provides only the access API for consumer applications; it's up to the implementation to provide additional modules within the framework that address functionalities such as property management, security, and connectivity for context providers to the model. The main benefits of using DCCI for pervasive smart-space applications are that it provides easy navigation, facilitates search and query for static and dynamic properties, and can support publish-subscribe mechanisms for local and remote context changes.

We've implemented DCCI as an extension to the Mozilla-based MicroB browser on the Maemo platform for Nokia's Linux-powered

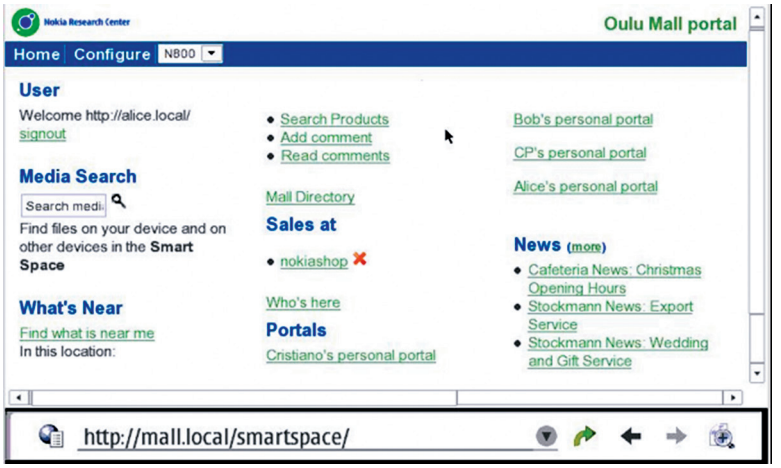


Figure 4. Composition of discovered services on a Web-based portal user interface (UI). The portal UI in this screenshot shows the shopping mall portal available in the local network with links to device portals from several users, shopping-mall-related news, and links to nearby services.

N810 Internet Tablet, which can easily be added to other Mozilla-based browsers.[8] This extension provides dynamic location information and static information such as screen size and width. The location parent node has two child nodes, location GPS and location indoor, which are linked with GPS and indoor location, respectively. The property change event is for sending notifications whenever the location changes. By using an event listener for the parent location node, an application (script) can monitor all location nodes below this node.

The extension also supports remote DCCI devices, through which users can create an ad hoc smart space over Bluetooth. When these remote devices are combined with the proxies shown in Figure 1, we can expose local context to non–DCCI-capable devices in the smart space. We use DCCI trees' hierarchical nature to construct a composite tree structure that includes local copies of DCCI trees of remote devices that are synchronized using events. Web applications see the composite smart-space context reflected in the browser's DCCI tree. In practice, each device's DCCI context tree could be different because of, for example, heterogeneous access rights or various needs of locally running applications.

Our analysis of the implementation has revealed a few drawbacks with the DCCI approach, however. First, because DCCI is derived from the DOM API, implementations must support all DOM element and node methods, including those intended for document manipulation. Only a subset of the API is relevant for context representation models. However, for compatibility reasons, an implementation must still support the full API. Second, a simplified event model would suffice as opposed to the full DOM event support. The DOM event mandates the implementation of event-capture, target, and bubble phases.

Third, DCCI provides an API to read context information provided by various context services. However, applications currently lack a standardized way to pass configuration information to these services, such as for configuring location-update frequency or other service parameters. Fourth, there is no standard ontology to which DCCI context trees conform. Such agreed-upon semantics would be necessary to ensure compatibility of DCCI context among applications. The W3C's Ubiquitous Web Applications (UWA) working group is developing an ontology to resolve this issue.

Finally, the lack of a security model for DCCI is inhibiting its widespread adoption. However, DCCI is intended for use with additional frameworks that can address this and other issues. For example, the resource access control mechanism discussed in this article can serve this purpose. We'd have to annotate the DCCI model with access rights, and we could also use DCCI to store membership assertions for the user.

Despite these current limitations, DCCI is a forward-looking approach that provides many advantages for future smart spaces, including its support for dynamic properties, access to data providers through a unified API, dynamic topologies, and a representation that separates syntax and semantics (via ontology).

Concept Evaluation and Experiences

To gain firsthand experience on the applicability of our Web-based smart-space approach in a practical setting, we developed a shopping mall smart-space demo. This has been running nonstop for several months in the Nokia showroom in Oulu, Finland.

The UI (Figure 4) presents users with a Web portal served from a mobile Web server that integrates a set of service portlets. These portlets represent distinct services in the system that users can find via our resource search solution. The integration of a Web-based UI with local service discovery ensures that each user receives a dynamic, up-to-date representation of the contents and services from devices participating in the smart space. The main services include a shop directory component, per-shop sales feeds, aggregated sales and news, a portal finder, a friend

finder, commenting, media sharing, and smart-space search. Users can come and go, and their presence in the smart place is reflected by their device portals appearing on the list of visible portals in the main screen of the portal running on their device.

Creating additional services is similar to creating ordinary Web applications. Security is integrated into the application framework (in our case, Python Django), and a Python library supports resource discovery for easy mashups.

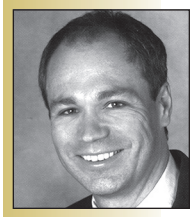
The main feedback we received from users that interacted with this system, which was running in Nokia premises for several months, was as follows:

- Location-aware maps and location-based services are good concepts that they'd like to see more of.
- They were generally able to accomplish simple tasks because using the system is similar to using a normal browser-based Web service.
- The UI's responsiveness is important, especially for blocking operations involving, for example, network discovery. This suggests that we should handle the discovery asynchronously, which is possible with our search-engine approach.

Generally, users were enthusiastic about our system, and we received many ideas for potential extensions and new use cases.

We've demonstrated the feasibility of providing pervasive smart-space services by implementing our prototype smart space. The user study we conducted confirms the acceptability of this approach to users, despite some usability issues. Our central claim is that these solutions can help achieve the vision of mass-market pervasive services. The REST-based approach advocated here can be easily applied to existing Web-based

the AUTHORS



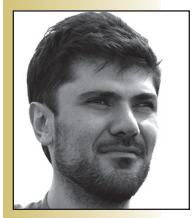
Christian Prehofer is an expert area manager at Fraunhofer ESK in Munich. He completed the work described in this article while he was a distinguished research leader at Nokia Research Center in Helsinki, Finland. His research focuses on adaptive systems for communication systems, especially mobile communication. Prehofer has a PhD and a habilitation in computer science from Technical University of Munich. Contact him at christian.prehofer@esk.fraunhofer.de.

Jilles van Gurp is a software architect at Nokia Gate 5 in Berlin. His research interests include object-oriented frameworks, software architecture, software product lines, and pervasive computing. Van Gurp has a PhD in computer science from the University of Groningen in the Netherlands. Contact him at jilles.vangurp@nokia.com.

Vlad Stirbu is a senior researcher at Nokia Research Center in Tampere, Finland, and he is pursuing a PhD in the Software Systems Department at the Tampere University of Technology. His research interests include GUI toolkits and service architectures. He has an MSc in computer science from Politehnica University of Bucharest. He's a member of IEEE. Contact him at vlad.stirbu@nokia.com.

Sailesh Sathish is a senior researcher at Nokia Research Center in Tampere, Finland. His research interests include context models and architectures, especially their relations to the mobile Web, as well as adaptive frameworks, user interest modeling, and Web technologies. Sathish has a licentiate of philosophy in computer science from Tampere University. Contact him at sailesh.sathish@nokia.com.

Pasi P. Liimatainen is a principal member of the engineering staff at Nokia Research Center in Tampere, Finland. His research interests include mobile Web technologies and user experience. Liimatainen has an MSc in information technology from the Tampere University of Technology. Contact him at pasi.p.liimatainen@nokia.com.



Cristiano di Flora is a software architect at Nokia Devices R&D. His research interests include software engineering for pervasive systems, context-aware computing, and Web technologies for mobile devices. Di Flora has a PhD in computer engineering from the Federico II University of Napoli. Contact him at cristiano.di-flora@nokia.com.

Sasu Tarkoma is a principal member of the research staff at Nokia Research Center in Helsinki, Finland, and a full professor in the Department of Computer Science at the University of Helsinki. His research interests include mobile computing and service architectures. Tarkoma has a PhD in computer science from the University of Helsinki. He's a member of IEEE and the ACM. Contact him at sasu.tarkoma@cs.helsinki.fi.

architectures and existing browsers on mobile devices. The key remaining challenges involve resolving usability issues such as those encountered in our study; establishing a widely used security and resource-finding solution, as proposed in this article; and spreading the use of technologies such as DCCI on the client side.

ACKNOWLEDGMENTS

A preliminary version of this work was presented at the Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures.[9]

REFERENCES

1. P. Debaty and D. Caswell, "Uniform Web Presence Architecture for People, Places, and Things," *IEEE Personal Comm.*, vol. 8, no. 4, 2001, pp. 46–51.
2. G. Lawton, "Web 2.0 Creates Security Challenges," *Computer*, vol. 40, no. 10, 2007, pp. 13–16.
3. C. di Flora and C. Prehofer, "Leveraging GIS Technologies for Web-Based Smart Places Services," *Proc. 6th IFIP WG 10.2 Int'l Workshop Software Technologies for Future Embedded and Ubiquitous Systems*, LNCS 5287, Springer, 2008, pp. 256–267.
4. F. Zhu, M.W. Mutka, and L.M. Ni, "Service Discovery in Pervasive Computing Environments," *IEEE Pervasive Computing*, vol. 4, no. 4, 2005, pp. 81–90.
5. B. Ferg et al., *OpenID Authentication 2.0—Final*, OpenID Community, Dec. 2007; http://openid.net/specs/openid-authentication-2_0.html.
6. M. Atwood et al., *OAuth Core 1.0*, OAuth Core Workgroup, Dec. 2007; http://oauth.net/core/1.0.
7. K. Waters et al., *Delivery Context: Client Interfaces (DCCI) 1.0: Accessing Static and Dynamic Delivery Context Properties*, World Wide Web Consortium (W3C) candidate recommendation, Dec. 2007; www.w3.org/TR/DPF.
8. A. Brodt, "Context-Aware Mashups for Mobile Devices," *Proc. 9th Int'l Conf. Web Information Systems Engineering* (WISE 08), LNCS 5175, Springer, 2008, pp. 280–291.
9. C. Prehofer, J. van Gurp, and C. di Flora, "Towards the Web as a Platform for Ubiquitous Applications in Smart Spaces," *Proc. UbiComp 2007 2nd Workshop Requirements and Solutions for Pervasive Software Infrastructures (RSPSI)*, Springer, 2007; www.igd.fhg.de/igd-a1/RSPSI2/papers/Ubicomp2007_RSPSI2_Prehofer.pdf.