RC24344 (W0709-023) September 7, 2007
Computer Science

IBM Research Report

Modeling and Managing Pervasive Computing Spaces Using RESTful Data Services

D. Coffman, S. McFaddin, C. Narayanaswami, D. Soroker
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

J. H. Han, H. K. Jang, J. H. Kim, J. K. Lee, M. C. Lee, Y. S. Moon, Y. S. Paik, J. W. Park
IBM Ubiquitous Computing Laboratory

Modeling and Managing Pervasive Computing Spaces using RESTful Data Services

D. Coffman, S. McFaddin, C. Narayanaswami, D. Soroker
IBM T.J. Watson Research Center
mcfaddin@us.ibm.com

J.H. Han, H.K. Jang, J.H. Kim, J.K Lee, M.C. Lee, Y.S. Moon, Y.S. Paik, J.W. Park
IBM Ubiquitous Computing Laboratory

Abstract

This paper advocates the use of RESTful data services in pervasive computing environments. REST is a restriction of web services to a simple protocol centered on a limited set of access operations against data resources. This paper outlines how REST may be used as a general design principle for pervasive computing environments. It additionally describes our implementation of a pervasive computing environment, called Celadon, which uses these design principles.

1. Introduction

Use of mobile devices has increased dramatically in the past decade. Voice, e-mail, and text messaging were the first applications to be widely adopted. Advocates project that "mobile commerce" functions, which include location and context sensitive functions such as shopping, payment and customized advertisement, will enjoy significant growth in coming years. In fact, recent surveys show that over 50 percent of all mobile device users would purchase devices that offer these functions and would pay a premium to both the device manufacturer and the service provider [1].

While this is an encouraging outlook for pervasive computing, its success hinges on the ability to address the associated issue of growing complexity in building mobile commerce and similar environments. Voice, e-mail, and text applications are generally based on fixed interfaces with well known service points: though the device may move around, the services do not. In contrast, the emergence of mobile commerce applications will require dynamic association of mobile devices and their users with a wide variety of other devices and services in local environments. Mobile commerce will require devices and environments to quickly recognize each other, do some business, and then say goodbye.

This transient usage pattern renders significant information modeling challenges. Web services offer a good substrate by providing a uniform model for discovery and usage of arbitrary object types and their interfaces. However, at the broader design level, substantial challenges remain. Notice that a single mobile commerce enabled device could travel through dozens of localized web services environments within a single day of usage. Each localized environment could offer dozens of different services, each bearing dozens of object types and their interfaces. This compounded complexity offers a challenge in the modeling, implementation and management of both the devices and the environments. In particular, the multitude of service interfaces should be managed by a simple and commonly accepted design pattern.

This paper presents an approach to managing information in pervasive computing domains based upon the design principles of "Representational State Transfer" (commonly called "REST") [3]. According to REST, which is becoming increasingly popular, a web service is viewed as a set of simple and common operations against a well understood set of resources. REST is a restriction of the more general remote procedure call (RPC) design style. With RPC, the application domain is analyzed into an arbitrary collection of object types, with each object type exposing customized (and possibly dissimilar) interfaces. The core weight of the design is carried by the interface analysis. In contrast, REST requires the same interface to be supported by each object type, throwing the weight of the design on the object-type decomposition itself.

The main contribution of this paper is a new methodology for designing pervasive computing spaces based on REST principles. Our methodology enables systematic construction of many different services that easily integrate into the space and are accessible by both mobile and non-mobile devices. We extract several general design principles for such an approach, which could be applied across many pervasive computing domains. We describe an extensive architecture and tooling platform we have built for modeling and implementing RESTful data services called Celadon. Celadon is presently being deployed in the Incheon Free Economic Zone (IFEZ), a "Ubiquitous City" [7] under construction in the suburbs of Seoul, Korea. We then offer an outline for applying this approach and architecture to other popular pervasive computing domains. We conclude with lessons learned and suggest several areas of future work.

2. Principles of REST

REST is not a strict discipline, nor is it embodied by a rigid standard or specification. Instead, it is a general design approach for the simplification of web services. Services which adhere to this design approach are said to be "RESTful" web services [4]. A wide variety of service types can be addressed through the REST approach, including graphics-oriented services (e.g.,, Yahoo maps [6]), data-oriented services (e.g., Amazon S3 [8]), and others.

In general, the REST design approach has the following three elements:

- *Resources*: The domain is analyzed and decomposed into a core set of fundamental resources. For data-oriented web services, this involves an identification of the core data types to be managed by the service.
- *Representations*: A form of representing and naming the core resources is identified. Representation for graphical services may be as visual diagrams. For data-oriented service, resources might be represented as documents with an agreed-upon structure. As far as naming, one scheme is to statically name all resources with fixed identifiers. Another approach is to offer resource identification through query–like representation.
- *Operations*. A small set of common operations against the resource types are identified; these comprise the common interface. For data-oriented services, the core operations might be simply: add, delete, update, and find.

This paper describes a data-oriented approach to modeling, managing and monitoring pervasive computing domains using REST principles. This approach is motivated by the zone-based services sub-domain of pervasive computing, which will be described in the next section. We will use this domain as an example to illustrate the data oriented REST approach.

3. Zone-based services: A data-driven pervasive computing domain

This section provides a detailed description of the zone-based services environment. This description will be used to motivate the design approach provided in subsequent sections of this paper.

In a zone-based services environment, businesses and other institutions offer localized services and interactions to mobile users. Zone based services are designed to be deployed in public spaces in which members of the public carry mobile devices, and transiently become users of the space. Examples of zone based services environments are shopping malls, sports stadiums, train stations, or hospitals.

Physically, a zone-based services environment is decomposed into different logical regions, called zones, cells, and locations. Zones are generally regions the size of a room – for example, a small store or a department within a larger store. Cells are generally individual areas within a zone in which a particular type of activity occurs, for example, a particular shopping aisle in a store. Locations are generally individual points – for example, a particular shelf within an aisle (cell), within a store (zone). The overall environment may be on the scale of a shopping mall, a hospital, or a city block.

As users enter the environment, they are registered with the system as members. Membership is a transient concept, lasting only for the duration of the user's visit. Various data about the member are collected, including credential data, which prove the identity and status of the user, as well as preference data, which describe the characteristics of the user and his intentions within the environment (e.g. a shopping list). As the member moves throughout the environment, his location is tracked. Based upon the member's location, correlations are triggered which match the characteristics of the member (e.g. categories within a shopping list) with the characteristics of the location (e.g. products offered along a particular aisle), and customized services and interactions are constructed and offered to the member. These services span a wide range, including alerts delivered to the member's device, customized advertisements that appear on displays at the member's location, interactions that allow the member to interact with the data provided in the environment (e.g. to query for restaurant locations and menus), and transactions that enable the member to participate in a business process (e.g. make payments, check inventories, schedule a home delivery). Figure 1 illustrates such an interaction.

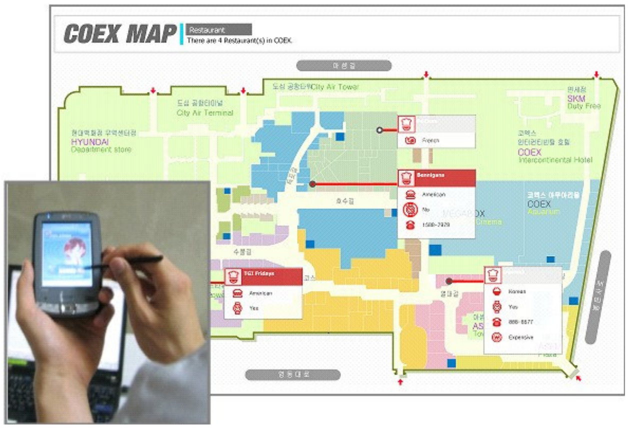


Figure 1: Restaurant search interaction between query service on member device and display service on facility device (large screen display).

We constructed a substantial system based on the Celadon architecture, for managing zone-based services environments. To support the types of scenarios described above, it was necessary to provide a distributed implementation that manages the various types of information such an environment entails.

This information must be made available across a wide variety of devices in the zone-based services environment. This includes the member's device, which carries member information and offers interaction services on the device's display, location sensing devices, which monitor user movement, business process servers where business logic decisions are made, as well as various displays and facility devices. Rather than adopting a different architectural approach for each device, we looked for a simple information management approach that would apply in a common way to all of the architectural elements.

The first step in constructing this system was to analyze the core information to be managed, regardless of its method of management and deployment on particular elements of the distributed architecture. Our analysis yielded the following categories of data:

Member Information: The environment must provide information about the members who have entered. This includes non only credentials and preferences, as provided by the member, but also data that is generated by the local environment based on reasoning about the needs of the member. Additionally, the member information must carry associations that correlate the member to various services offered to and consumed by the member.

Service Information: The environment must provide information about the services offered within a particular zone. In a zone-based services domain, the concept of service encompasses a spectrum of offerings ranging from physical services, such as product information offered at a retail shelf location, to functional services, such as a payment web service. Additionally, the service information must include the underlying data service offered by the environment itself. Service information must consist of rich data types, supporting arbitrary attributes and parameters, and substructures. The service information must be active, for example, carrying status information about service availability (e.g. it should reflect whether a particular station within the zone is occupied).

Location Information: The environment must provide information which correlates zone members to their physical and logical locations. The location information must accommodate the presence of the member in possibly overlapping zones, cells, and location points. The location information must be active, meaning that as the member moves physically throughout the zone, software components can sense the changes in zone, cell, and location.

Interaction Information: The environment must provide information which correlates members to services and interactions offered throughout the environment. It must be possible to suggest one or more interactions to the member through this data. When an interaction is accepted, either explicitly or implicitly, the interaction must capture the correlation of the member with the local service underlying the interaction such as a physical display or payment station.

4. Applying the REST design approach to data-driven pervasive computing domains

This section describes our approach for applying the REST design principles to the zone based services domain, and provides guidelines for applying this approach to other domains.

Resource Analysis: Our approach is based on a three level decomposition. The first level is a decomposition of the data space into distinct *data services*. These are macroscopic aggregations of the overall resources of the domain. The decomposition used in our design corresponds to the above categories, yielding four core data services. The second level divides each data service into a set of *resource types* – these are the distinct data types that a data service is responsible for managing. Each data service must be capable of carrying a multiplicity of instances of each of its supported resource types. At the third level each resource type is described by a data definition which defines the individual *resource elements* – these are the data fields and members that define a resource as a composite.

Representational Analysis: In most cases the base representation will be determined by the infrastructure – we have chosen to represent resources as (1) XSD (XML Schema) type definitions at tooling time, (2) Java and JavaScript data types at runtime, (3) XML documents during interchange, with (4) HTTP used as a transport. Other base representations are possible – for example, a C-based runtime may be used, and SOAP may be used for transport. Additionally, the representation must provide for a naming and access model – we have chosen to provide a query-oriented access model. This means that resources are retrieved by pattern matching against templates, rather than being retrieved by names. This approach is important in pervasive computing spaces, whose transient and ad-hoc natures make fixed naming schemes impractical.

Operations Analysis: The third step in the approach is to establish a reasonable set of operations against the resource set, and which will range over the representation as its operands. We have chosen a very simple model based on four operations – add, delete, update, and find. Additionally it is important to provide for events – these are handled by providing a corresponding event to each of the basic operations, along with appropriate filtering and subscription capabilities.

The following table summarizes the resource, representation, and operations analysis for the zone based services scenario.

Table 1. REST analysis for zone-based services.

Resource Type	**Resource Elements**	**Meaning of Operations**
Member Information Service		
Member Information Records Basic information about a member of the zone, generally submitted by the member themselves.	Member ID field Credential list (e.g. X.509 identity, shopper ID) Status field (e.g. "busy", "waiting", "inactive")	Add: member arrival Delete: member exit Update: member status changed (e.g. zone activity) Find: look for member by ID, look for all "busy" members
Role Information Records Each record correlates a logical role to a member – these records are assigned by decision making agents in the zone environment.	Member ID field Role ID field (e.g. "shopper", "premium shopper", "payer")	Add: zone assigns role to member Delete: zone withdraws role Update: zone changes role Find: look for all roles assigned to a member (search by ID), look for all members having a given role ("premium shopper", etc.)

Service Information Service		
Service Information Records Each record registers a service in the environment – services can be registered by visiting members or by the zone itself through distinct member IDs. Services can be dynamically added, withdrawn, and have their status changed.	Service ID field Member ID field Service Type field (e.g. "data service", "display service") Attribute List Service Status field ("active", "available", "unavailable)	Add: member or facility device registers service Delete: member previously registering service leaves Update: service status changes (e.g. from "available" to "active")
Location Data Service		
Location Information Records Each record correlates a logical location with a zone member – changes in these records may be monitored to trigger business processes.	Member ID field Zone ID field Cell ID field Location ID field X,Y,Z fields (optional)	Add: member enters a zone, cell, or point location. Delete: member exits a zone, cell, or point location. Update: member moves.
Access Point Information Record Used for administration – this record tracks the location and configuration of access points and is used by triangulation brokers to locate members through signal strengths.	SSID field – identifies the access point X, Y, and Z fields – base location of access point for triangulation. RSSI field – relative signal strength of access point	Add – used at zone configuration time to declare an access point Delete – removes an access point Update – access point was moved or reconfigured.
Interaction Information Service		
Interaction Information Record Each record correlates a zone member to a set of services that aggregate into an interaction – e.g. a display service in a context driven advertising scenario. Note interactions may be suggested by the system and not activated.	Interaction ID field Member ID field – the member for whom the interaction is created. Service ID list -- the set of services aggregated into the interaction. Status field (e.g. "suggested", "active", "idle")	Add – an interaction is suggested by high level business logic in the zone. Delete – an active interaction is completed, or a suggested interaction is withdrawn (e.g. user moves away from interactive advertising display). Update – status of interaction changes (e.g. from "suggested" to "active" due to user acceptance)

There is art to the decomposition process. In general the various components of the decomposition will be distributed across the distributed architecture in a way that will have ramifications both for the logic and performance of any implementation. For example, instances of data services will operate on individual platform instances in the design. In many cases these instances will be disconnected and also portable across different environments. Each data service should comprise a rational collection of information that makes sense for a device or server to carry. For example, an instance of a service information data service could operate on a server as a directory mechanism. An instance of a service information service could also be deployed on a device as a way of storing information about the local services on board the device. (Note that the underlying implementations may be radically different – a SQL database in the first case and a file based XML store in the second – but the data decomposition and interfaces would be common. Each resource type should be defined in a way that it captures the concept of a locus of data where a reasonable amount of interrelated changes would be occurring. This allows the reporting of events to be simplified to simple notifications that a particular resource has changed, without requiring further analysis about which part of the resource changed. If resource definitions are too "fat" then large amounts of update data would need to be transferred whenever a minor change is made. Resource definitions which are too "thin" would result in data services with a large number of microscopic resources.

5. Design principles for RESTful data service implementations

In addition to the above REST design principles for data analysis, it is also important to establish design principles for runtime and tooling implementations for pervasive computing domains. This section identifies several key implementation design principles which we have extracted from our work in the zone based services domain.

5.1 Ease and Ubiquity of Access

The principal motivation for this work is that the data service be simple and easy to use. The data service must be easily accessible from all of the software platforms in the pervasive computing domain in which the data service operates. It must make use of existing representations and protocols. In our zone based services domain, we have used simple HTTP/XML for network interchanges. Additionally, we have provided runtime support for the J2SE and J2ME Java environments as well as AJAX-style environments (consoles, interactive displays and kiosks, and script-drive device environments).

5.2 Ease of Implementation

It should be simple and easy to implement and operate a data service for a pervasive computing environment. It should be easy to add new data artifacts to the environment as data models change. It should be possible to implement data services on devices other than servers. It should also be possible to integrate the pervasive data services with commonly available tooling platforms. Toward this goal we have also implemented a substantial Eclipse-based tooling environment which generates object implementations, service runtimes, client stubs, editor frameworks, translators, parsers and other components for the target environments listed above.

5.3 Rich Data

The concept of simplicity, advocated in the above design principle, should not limit the richness of the data carried by the data space. The types of data offered by the space must be allowed to be arbitrarily rich – it is only the mode of access that should be simplified. Thus, for example, implementations should be reasonably agnostic about the depth and structure of the data records carried in the representation.

5.4 Self Describing

A pervasive data service should be self describing. This means that it must be possible for an application to contact a pervasive data service and ask it for the types of data it carries. Based only on this knowledge it should be possible for an application to fully interact with all of the capabilities of the data service, and to handle all of the data types supported by the service. The self-description process should use the core protocol of the service itself and not require additional protocol implementations simply for discovery.

5.5 Event Driven

Pervasive computing environments are highly dynamic and generate a lot of events. This is true of both sensor-oriented environments and human-oriented environments. The pervasive data space must be able to accommodate sensing elements in a physical layer which wish to make frequent changes to the data space. Monitoring components must be able to listen for such changes and to periodically receive events about changes.

5.6 Scalability

The pervasive data space must be scalable. In particular it must scale both with the number of artifacts modeled (i.e., the number of data records) and with the intensity of changes to those artifacts (i.e., the number of events). This means, in particular, that applications should be able to filter their view of a data space both against the types of operations against the data space and against the content of the modeled artifacts. For example, it should be possible for an application to isolate and monitor the movements of a single user of a data space in an environment where there are hundreds of other users. Event processing and other computational resources should be consumed only in proportion to the activities of that user. In short, reasonably powerful filtering capabilities must be present.

6. Celadon Data Service Implementation

This section describes the RESTful data services implementation offered by the Celadon environment. Celadon provides only a design pattern for a set of operations (methods) and event objects. Additionally Celadon provides tool-generated runtime implementations. Celadon may be carried by any underlying service and event delivery technologies.

Celadon data services are based on a set of simple transactional operations and a set of event types.

Each Celadon operation is comprised of (1) an operation name, (2) an operand datum, and (3) a result datum. Celadon operation names are constructed from the core operations defined in the data analysis above plus the name of the resource type (e.g. "findMemberInformationRecord"). The operand describes the resources affected by the operation. As noted above, we use query patterns, rather than rigid names, in Celadon operands. Query patterns are given by a single instance of the type of resource being queried for (e.g. a MemberInformationRecord) which is used as a template. Fields which are set in the template are used to match corresponding fields in candidate resources on the data service. For example, to find all member information records having a given memberID, a caller would instantiate a blank member information record as a template, set the memberID field, and pass this instance as the operand of a findMemberInformationRecord object.

Celadon operations are atomic and have no side effects, meaning that they are considered to be committed and completed upon return. Celadon operations may be accessed either through a programmatic API (e.g., a Java API) or through HTTP transactions. For Java access, each data services is represented by a service object having a set of methods whose names match the operation names described below. Each method has zero or one parameters (the operand datum) and either a void return value, or a typed return value matching the type of the operand. The HTTP access is via a GET or POST operation against an HTTP service registered at a network location given by a `(protocol, address, port, path)` tuple. The operation name is extracted from a parameter value called "`operation`" which may be provided in either the POST data or in the path suffix in the case of GET. For example, the URL

`http://celadon.ibm.com:9080/MemberInformationService?operation=findMemberInformationRecord`

would be used to invoke the "`findMemberInformationRecord`" on a data service operating at the network coordinates: `("http","celadon.ibm.com","9080","MemberInformationServiceServlet").`

6.1 Self Description

Celadon data services described themselves through XML schema records. The following table describes the operation which accesses the schema:

Table 2. The Celadon self description operation

Operation Name	`getDataSchema`
Operand Datum	None
Result Datum	`<xsd:schema>` element

The returned schema declares the supported data types as instances of top level `<xsd:element>` descriptions. The name of the element is used as the record name expected in all other operations (e.g. "`XYZRecord`"), and the type definition referenced by the element defines the structure of the data type carried by the data service. Note that this approach allows a data service to carry any number of simultaneous data types and to implicitly define the operations that may be carried out on those data types, as described below.

6.2 Core Operations

For each data type supported by a data service, Celadon provides four fundamental access operations. The operation names key on the data record name given by the schema provided above (e.g. "`XYZRecord`").

Table 3. Core Celadon operations

Operation Name/ Meaning	Operand Datum	Result Datum
findXYZRecord Filtered Lookup	Pattern Record	List of records
addXYZRecord Record Creation	New Record	None
deleteXYZRecord	Pattern	None

Record Deletion	Record	
updateXYZRecord Record Alteration	Updated Record	None

6.3 Event Subscription

Celadon provides events corresponding to each of the above core operations. For example, corresponding to the addMemberInformationRecord operation is a memberInformationRecordAdded event type. Subscribers are added through similarly named operations. A subscription is made by registering a template object (see the discussion of the find method above) against which events are filtered. Subscribers will only receive events for resources on the data service matching the template object.

Table 4. Celadon event subscription operations

Operation Name/ Meaning	Operand Datum	Result Datum
addXYZRecordListener Subscriber registration	Listener ID plus Pattern Record	None
deleteXYZRecordListener Subscriber deregistration	Listener ID	None
getXYZRecordListenerEvents Event retrieval – returns list filtered on pattern record	Listener ID	List of events

6.4 Runtime environment

The broader Celadon architecture integrates the data services into an overall services-oriented eco-system for zone based services environment. The overall environment is illustrated in Figure 2.

The Celadon environment integrates the core REST data services described above with a variety of other data services and components. This may include application oriented data services which are also designed using the REST analysis and generated using the Celadon data tools (see below). For example, the shopping example described above relies upon REST-based data services to manage restaurant locations as well as product information stores. In other applications we have used REST-based data services to support a large RFID item association service, which associates RFID tag values with item-wise product descriptions.

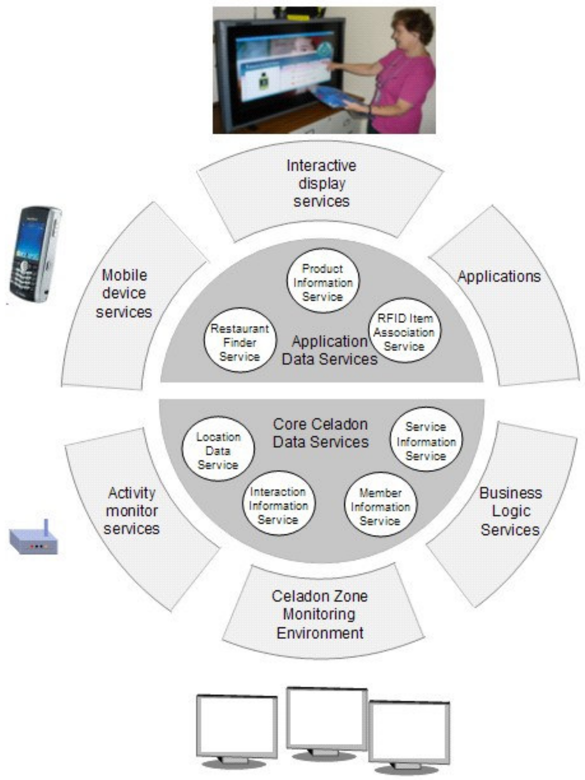

Interactive display services
Mobile device services
Applications
Product Information Service
Restaurant Finder Service
Application Data Services
RFID Item Association Service
Core Celadon Data Services
Location Data Service
Service Information Service
Interaction Information Service
Member Information Service
Activity monitor services
Business Logic Services
Celadon Zone Monitoring Environment

Figure 2: Integration of data services into the overall Celadon architecture for zone-based services

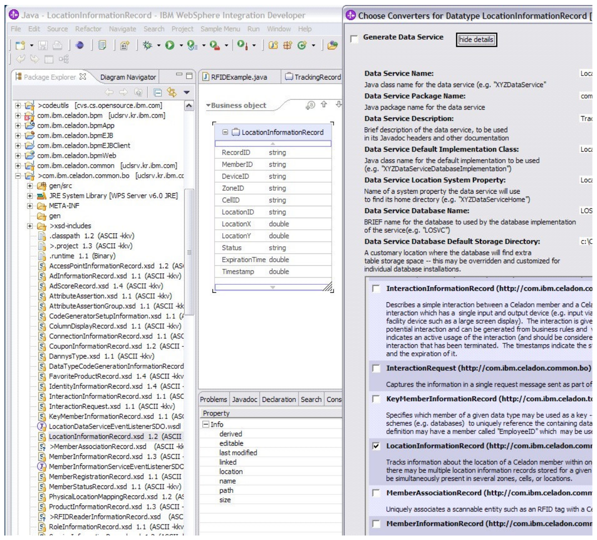
The Celadon environment also integrates data services with services on various devices and display applications. Additionally Celadon provides a management and monitoring environment, which is driven by a set of consoles which monitor selected working sets of resource records in the core data services layer. Additionally Celadon provides support for low level activity brokering component (e.g. a signal strength triangulator which interacts with the location information service) as well as high level business process decision components (e.g. ontology-based classifiers which interact with the member information service to assign logical roles to zone members). A full description of the broader Celadon environment is provided in a separate paper [5].

7. Tooling for Pervasive Data Services

This section describes a various classes of tools which may be applied to RESTful data services.

7.1 Celadon Data Tools.

Celadon provides a full service tooling environment for RESTful data services in pervasive computing environments. The tool enables a human designer to carry out the resource analysis at the core of the REST design approach. This analysis occurs in a modeling session in which a domain expert analyzes the core resources of a particular domain which is to be supported by RESTful data services. The domain expert does not have to be familiar with any of the underlying resource representations, nor an expert in web services or


Java - LocationInformationRecord - IBM WebSphere Integration Developer
File Edit Source Refactor Navigate Search Project Sample Menu Run Window Help
Package Explorer
Diagram Navigator
RFIDExample.java
TrackingRecord
Business object
Problems
Javadoc
Declaration
Search
Cons
Choose Converters for Datatype LocationInformationRecord [
hide details

```
>codeutils  [cvs.cs.opensource.ibm.com]
com.ibm.celadon.bpm  [uclsrv.kr.ibm.com]
com.ibm.celadon.bpmApp
com.ibm.celadon.bpmEJB
com.ibm.celadon.bpmEJBClient
com.ibm.celadon.bpmWeb
com.ibm.celadon.common  [uclsrv.kr.ibm.com]
>com.ibm.celadon.common.bo  [uclsrv.kr.ibm.cc
   gen/src
   JRE System Library [WPS Server v6.0 JRE]
   META-INF
   gen
   >xsd-includes
   .classpath  1.2  (ASCII -kkv)
   >.project  1.3  (ASCII -kkv)
   .runtime  1.1  (Binary)
   AccessPointInformationRecord.xsd  1.2  (AS
   AdInformationRecord.xsd  1.1  (ASCII -kkv)
   AdScoreRecord.xsd  1.4  (ASCII -kkv)
   AttributeAssertion.xsd  1.1  (ASCII -kkv)
   AttributeAssertionGroup.xsd  1.1  (ASCII -kk
   CodeGeneratorSetupInformation.xsd  1.1  (A
   ColumnDisplayRecord.xsd  1.1  (ASCII -kkv)
   ConnectionInformationRecord.xsd  1.1  (ASC
   CouponInformationRecord.xsd  1.2  (ASCII -
   DannysType.xsd  1.1  (ASCII -kkv)
   DataTypeCodeGenerationInformationRecord
   FavoriteProductRecord.xsd  1.4  (ASCII -kkv
   IdentityInformationRecord.xsd  1.4  (ASCII -
   InteractionInformationRecord.xsd  1.1  (ASC
   InteractionRequest.xsd  1.1  (ASCII -kkv)
   KeyMemberInformationRecord.xsd  1.1  (ASC
   LocationDataServiceEventListenerSDO.wsdl
   LocationInformationRecord.xsd  1.2  (ASCII
   >MemberAssociationRecord.xsd  (ASCII -kk
   MemberInformationRecord.xsd  1.3  (ASCII -
   MemberInformationServiceEventListenerSDC
   MemberRegistrationRecord.xsd  1.1  (ASCII
   MemberStatusRecord.xsd  1.1  (ASCII -kkv)
   PhysicalLocationMappingRecord.xsd  1.2  (AS
   ProductInformationRecord.xsd  1.3  (ASCII -
   >RFIDReaderInformationRecord.xsd  (ASC
   RoleInformationRecord.xsd  1.1  (ASCII -kkv
```

LocationInformationRecord	
RecordID	string
MemberID	string
DeviceID	string
ZoneID	string
CellID	string
LocationID	string
LocationX	double
LocationY	double
Status	string
ExpirationTime	double
Timestamp	double

```
Property
Info
   derived
   editable
   last modified
   linked
   location
   name
   path
   size
```

☐ Generate Data Service

Data Service Name: Loca
Java class name for the data service (e.g. "XYZDataService"
Data Service Package Name: com
Java package name for the data service
Data Service Description: Trac
Brief description of the data service, to be used in its Javadoc headers and other documentation
Data Service Default Implementation Class: Loca
Java class name for the default implementation to be used (e.g. "XYZDataServiceDatabaseImplementation")
Data Service Location System Property: Loca
Name of a system property the data service will use to find its home directory (e.g. "XYZDataServiceHome")
Data Service Database Name: LOS
BRIEF name for the database to used by the database implementation of the service(e.g. "LOSVC")
Data Service Database Default Storage Directory: c:\C
A customary location where the database will find extra table storage space -- this may be overridden and customized for individual database installations.

☐ **InteractionInformationRecord (http://com.ibm.celadon.co**
Describes a simple interaction between a Celadon member and a Cela
interaction which has a single input and output device (e.g. input via
facility device such as a large screen display). The interaction is give
potential interaction and can be generated from business rules and
indicates an active usage of the interaction (and should be considere
interaction that has been terminated. The timestamps indicate the s
and the expiration of it.

☐ **InteractionRequest (http://com.ibm.celadon.common.bo)**
Captures the information in a single request message sent as part of

☐ **KeyMemberInformationRecord (http://com.ibm.celadon.t**
Specifies which member of a given data type may be used as a key -
schemes (e.g. databases) to uniquely reference the containing data
definition may have a member called "EmployeeID" which may be use

✓ **LocationInformationRecord (http://com.ibm.celadon.comr**
Tracks information about the location of a Celadon member within on
there may be multiple location information records stored for a given
be simultaneously present in several zones, cells, or locations.

☐ **MemberAssociationRecord (http://com.ibm.celadon.comm**
Uniquely associates a scannable entity such as an RFID tag with a Ce

☐ **MemberInformationRecord (http://com.ibm.celadon.comr**

Figure 3: Eclipse-based Celadon data tools for REST based data services

mobile computing platforms.

The tool operates as a plug-in software suite which is added to an Eclipse programming environment. The tool follows the data driven approach, beginning with data definitions for the core domain resources. The data types are represented by XML schema definitions (XSD) and may be defined using any standard schema editor. The schema editor defines the resource elements.

Once the resources are defined, the tool aggregates those resources into a data service definition. The user provides a name, description, and namespace for the data service and then selects resource types which are to be supported by the data service. The resource types are selected from a working set which is computed via introspection across the user's Eclipse workspace.

The tool generates a variety of runtime classes which define and support the data service across a variety of platforms. The tool offers various implementation options. In particular, the tool generates data service implementations and distributes object representations and client stubs across different platforms. The platforms targeted by the current implementation of the tools are:

- J2SE platform for the runtime for the core data services.
- J2ME platform on mobile devices.
- AJAX platform for consoles, facility devices (kiosks and interactive advertising displays) and some mobile devices.
- J2EE platform for business process management.

Output options for the tool include:

- Java interface definitions and object implementations.
- A high performance SQL-data service implementation.
- A J2ME client stub which J2ME devices may use to interact with remote data services (via HTTP/XML).
- A JavaScript client stub for AJAX based applications.
- A JavaScript client editor which manages working sets of resource instances on an AJAX clients.

A sample screen using the Celadon data tools is seen in Figure 3. In this screen the user is defining a data service (in this case the core "Location Data Service" described above). The part of the screen at the left is a data type editor which defines the elements of a "LocationInformationRecord" resource data type. The part of the screen at the right is capturing the various generating options for the data service at the top and is performing the resource selection for the data service at the bottom

7.2 Domain Oriented Tools

Celadon also provides a variety of domain centric tools for zone configuration and monitoring. These tools operate by attaching to running instances of data services and generating records that configure the state of a zone environment. The tool seen in Figure 4 is a zone configurator - essentially a client application which integrates with the location data service to register, edit, update the data records which describe the geometries of the underlying zones, cells, and locations in the zone and configure the underlying access point in the environment.

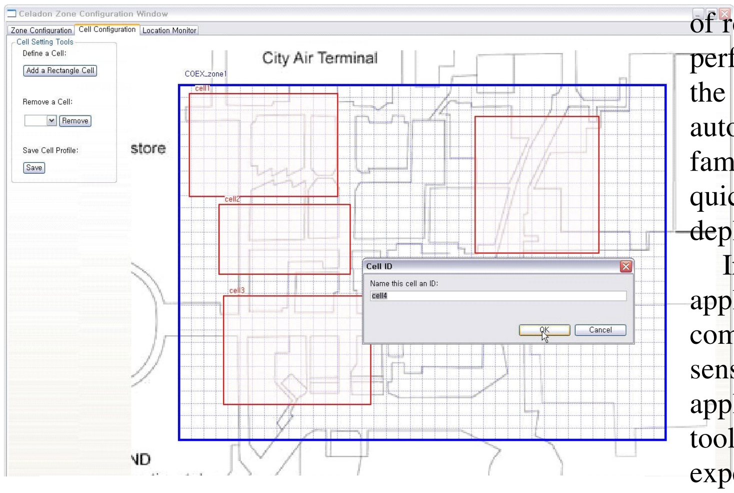


Figure 4: Zone configuration tool

8 Related Work

The REST principles date back to Roy Fielding's Ph.D. thesis from 2000 [3], and have gained popularity in the Web (e.g., [6], [8]). However, the application of REST as a design principle for pervasive spaces is new. Drytkiewicz et. al. [9] describe a REST-based protocol implementation for pervasive devices. Their focus is on enabling REST functionality on very limited devices, rather than applying REST principles for simplifying the design of pervasive spaces. Other approaches for improving efficiency of web service interaction on mobile devices include optimizations of various parts of the "traditional" services stack, such as SOAP messaging [12]. Our pervasive space architecture is loosely coupled and based on web services. Approaches for more tightly-coupled systems have included Active Spaces [11] and Tuple Spaces [10].

9 Summary, Conclusions, and Future Work

This paper has presented a methodology and design approach for using REST as an organizing principle in pervasive computing. We have implemented a real-life metropolitan zone-based services environment with this approach and have begun its deployment on a trial basis.

We have drawn several lessons from our experience. While we have found the implementation to be simple and scaleable across a wide variety of devices and clients, proper balancing of resource elements and data services is key to high performance. Also we have found that use of “off the shelf” data modeling tools coupled with automatic data service generators enable developers familiar with basic REST design principles to quickly create and implement new data services and deploy these throughout a zone-based environment.

In the future, research should be conducted into applying these design principles to other pervasive computing domains such as telematics, health care, sensor and actuator environments, and RFID application. Another focus area could be extending tooling support for REST-based domains to non-experts. Additionally further investigation into run-time options is needed. We have taken an approach which relies upon automatic generation of data services, clients and other handlers. We are also investigating a dynamic approach in which client modules work with the self-describing data and provide handling of data records at run-time. Event roll-up and security of RESTful services should be investigated. Adapters for other run times such as Service Data Objects (SDO) should be pursued.

10 Acknowledgments

This work is partially supported by the Institute of Information Technology Assessment and Ministry of Information and Communications of Republic of Korea.

11 References

1. M. Megna, “Mobile Commerce: Tapping the Untethered Market”, *Ecommerce-Guide.Com*, http://www.ecommerce-guide.com/solutions/secure_pay/article.php/3659656, February, 2007.
2. M. C Rosu, et al., “Celadon: A Novel Architecture for Symbiotic Computing,” In *Proc. International Symposium on Ubiquitous Computing Systems* (UCS) 2006.
3. R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, Ph.D. Dissertation, *UC Irvine,* 2000.
4. L. Richardson and S. Ruby, *RESTful Web Services*, O’Reilly Media, Inc, 2007.
5. Celadon: Pervasive Computing Meets Business in Public Spaces, in preparation.

6. http://developer.yahoo.com/**maps**/rest/V1/geocode.html
7. http://ucityeng.**ifez**.go.kr/overview/u_outline.asp
8. http://docs.amazonwebservices.com/AmazonS3/2006-03-01/RESTAPI.html
9. W. Drytkiewicz et. al. "pREST: a REST-based protocol for pervasive systems", IEEE International Conference on Mobile Ad-hoc and Sensor Systems, 2004, pp. 340- 348
10. B. Johanson and A. Fox. “Tuplespace-based Coordination Infrastructures for Interactive Workspaces,” Journal of Systems & Software, Spring 2003.
11. M. Roman et. al., "A middleware infrastructure for active spaces" IEEE Pervasive Computing, 2002, v. 1 pp. 74-83
12. Rosu, M., "A-SOAP: Adaptive SOAP Message Processing and Compression", ICWS 2007. pp. 200-207.