Towards RESTful Communications in Self-managing Pervasive Systems

Meherun Nesa Lucky, Christina Tziviskou, and Flavio De Paoli

Dipartimento di Informatica, Sistemistica e Comunicazione, Università di Milano-Bicocca
Viale Sarca 336/14, 20126, Milan, Italy
{meherun.lucky,christina.tziviskou,depaoli}@disco.unimib.it

Abstract. The presence of heterogeneous communication protocols and interfaces prevents from giving self-managing capabilities to service systems. This paper proposes the exploitation of the widely adopted HTTP protocol to create a shared platform that fosters the definition of services that can be easily integrated and controlled. Such services will be provided with RESTful interface and interaction style to gather data and control behavior of sensors to support the development of sensor services integrated with other services in pervasive systems.

Keywords: RESTful services, Sensors, Web Application, pervasive computing.

1 Introduction

Over the last decade, the Web has grown from a large-scale hypermedia application for publishing and discovering documents (i.e., Web pages) into a programmable medium for sharing data and accessing remote software components delivered as a service. The need of global availability and sharing of huge amount of information through various kinds of heterogeneous devices and services has changed the reference scenario for the development of Web scale applications. The consequent growing complexity and increasing request of adaptive services has made manual management impractical. A possible answer toward self-management is to make services smarter by achieving awareness of the target things' or the applications' physical environment or situations to respond proactively and intelligently. To tackle the problem a first effort should be the definition of a common protocol to foster automated interoperability.

As a matter of facts, the HTTP protocol has been used as a universal mean for tunneling messages in business-to-business scenarios, but quite often to support additional protocols, such as WSDL/SOAP for Web Services or OGC/SOS/SPS for sensors, that are usually not interoperable. In this paper, we investigate the correct and complete use of the HTTP protocol to publish, manage, and operate services on the Web by fully exploiting the REST (REpresentational State Transfer) principles [5]. Today, more and more services published on the Web are claiming to be designed using REST, but actually missing some of the features, like the hypermedia control, which is crucial to automate self-management operations.

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 263–274, 2013.

The rest of the paper discusses on integrating data from sensors and other services, which will be gathered by invoking different services through standardized interfaces by the Self-manageable Pervasive System to fulfill its task. To investigate the issue, we discuss two scenarios that differentiate between user control activities, where users are asked to invoke different types of services to fulfill a composite task, and self-managing pervasive system, where users invoke a single composite service without further interaction.

2 Motivations

Nowadays, the number and variety of available services on the Web enable for the definition of sophisticated composed services, which can take great advantage from pervasive computing and sensor Web to give users seamless assistance in daily activities. In pervasive computing environments, different devices often use different technologies, which make the communication complex and sometimes even troublesome. Sensor Web has a central role in addressing new sources of information that need to be dynamically integrated into systems. The Sensor Web Enablement architecture of the Open Geospatial Consortium (OGC)[2] has been developed to enable flexible integration of sensors into any type of software system using standards: the Observations & Measurements and Sensor Model Language specifications define the exchanged sensor data; the Sensor Observation Service (SOS) gives pull-based access to observation data [3]; the Sensor Planning Service (SPS) tasks sensors [15].

We want to point out, through the discussion of a use case, the problems that arise from the diversity of the involved services, and show how the communication burden can be transferred from the user to self-managing pervasive systems. Then, we investigate the advantages of the RESTful approach as a mean for providing a common interface for services communication and control.

2.1 Use Case Scenario

We consider two versions of a scenario in which a user, let's say John, wants to go to watch a movie looks for route directions to available parking spaces close to the cinema hall. In the first version, the user is asked to interact with different Web Services of different types, such as Web Form-based, SOAP, API and RESTful Web services. As we have included sensors in our scenario, different OGC compliant services such as SOS and SPS are invoked. In the second version, a self-managing composite service minimizes the user interaction by transferring the control from the user to the service. Self-management will be achieved by providing sensors and services with RESTful interfaces to make interaction and control similar to the one already in use on the Web and therefore already familiar to both users and developers.

Use Case Scenario with User Control. As depicted in Fig. 1 (a), first John searches for movies by filling up the Web form of the Cinema Portal, and gets the list of movies meeting his search criteria, with the corresponding cinema hall name. Then, John chooses a cinema hall and invokes the Location Service API to get the address. Next, he invokes the Parking System, with RESTful interface, to get information about the available parking spaces close to the selected cinema hall. The Parking System interacts with the SOS and SPS services to get the observations from the video cameras and location sensors installed in the parking lots, and makes decisions about the available parking spaces. Finally, John chooses one parking lot from the list of the available ones, indicates his current position, and invokes the Route Service, through SOAP messages, to get directions on how to reach his destination.

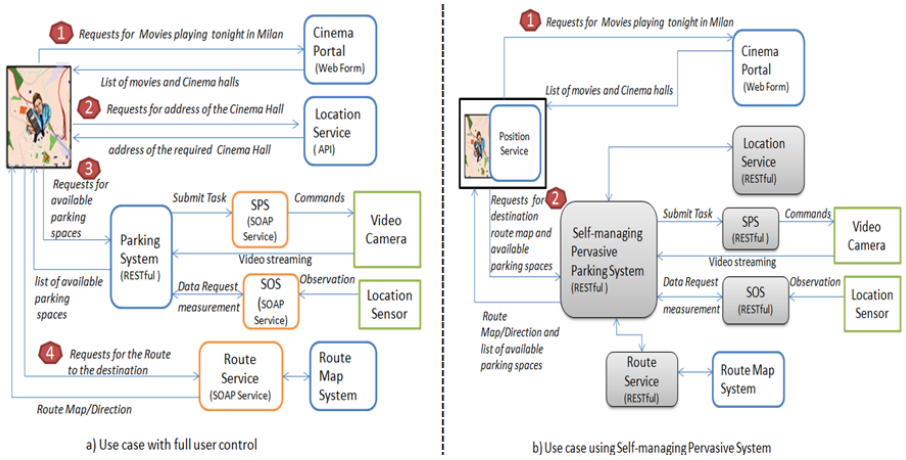

Fig. 1. Use case scenario

Use Case Scenario Based on a Self-managing Approach. In this version, the scenario has been changed to minimize the user control and add self-manageability to the system. The user invokes only two services: the Web Form-based Cinema Portal and the Self-managing Pervasive Parking System. Further, as shown in Fig. 1 (b), the user has the Position Service in his mobile, which is able to get automatically the position of the user and deliver it to other services. At first, John interacts with the Cinema Portal to get the list of movies of his interest, as before. Then, he selects a cinema hall and invokes the Parking System to get all the information needed to reach the destination. The Parking System interacts with different services: (i) invokes the Location Service to get the location of the selected cinema hall, (ii) invokes SOS and SPS to get observations about parking spaces, and decides which are the available parking spaces near to the cinema hall, (iii) gets the route direction for the nearest available parking spaces by interacting with the Position and the Route services, and (iv) delivers the cinema hall route direction to the user with maps.

2.2 Common Interface for Services Interactions: The RESTful Approach

In order to increase the system interoperability, we have designed the Self-managing Pervasive Parking System according to the Resource-Oriented Architecture that is devoted to manage distributed, heterogeneous resources in which client applications interact directly with the resources, by following the REST principles[5]:

1. Resources should be identified properly using URIs, so that each resource is uniquely addressable.
2. Uniform interfaces should be provided through the use of a standard application-level protocol. In this way, the operations to be applied on resources are external and they have well known semantics [12].
3. Resources are manipulated through their representations, since clients and servers exchange self-descriptive messages with each another. A resource can have multiple representations that follow a standardized format or media type and can be negotiated with the Web server. Representations convey the state of the client's interaction within the application and contain hyperlinks that allow clients to discover other resources or change the state of the current resource.
4. Interactions are stateless since servers only record and manage the state of the resources they expose, i.e., client sessions are not maintained on the server. This increases the decoupling between client and server.
5. Hypermedia is the engine of application state, i.e., the application state is build following hyperlinks according to the navigation paradigm. Therefore, the application state is not known a priori, but it is built based on user navigation.

Further, we expose data produced by the services according to the Linked Data paradigm. Linked Data Design defines[1] the following rules for exposing structured data on the Web: (i) use URIs to identify data as names, (ii) use HTTP to look up those names, (iii) provide useful information about URIs using standards, and (iv) include links to other URIs, so that they can discover more things.

Most of the technologies used in the first version of our use case (Fig. 1 (a)) do not follow the REST principles. Consequently, they compromise the interoperability of services and do not facilitate computer-to-computer interactions in the context of self-managing pervasive systems. In particular, the Route, SOS and SPS services communicate through SOAP messages. Thus, the HTTP methods are used as transport protocol while the application protocol is domain specific and the operations invoked by the user lay on the message envelope. Such communication pattern tunnels all the requests to a single URI that identifies an endpoint. HTTP GET and POST are the most-in-use methods but their semantics are not maintained, i.e. GET is used to invoke operations on server side that modify resources state, and therefore, it is not possible to optimize the network traffic by using caching mechanisms. Different is the case of the Web form-based Cinema Portal. The user interacts with different URIs via an html form, using again GET or POST possibly with a different semantics: URIs encapsulate server-side information, like operation names and parameters, revealing implementation details to the user. Such an approach enforces

[1] `http://www.w3.org/DesignIssues/LinkedData.html`

the coupling between the client and the server: if the server implementation changes, then the old URIs become invalid (operation and/or parameter names may change). The Location service publishes an API but interactions with the service are not hypermedia-driven since resources representation does not contain hypermedia controls and the user advances to the next state using some out-of-band information .

The pervasive computing system is expected to be seamlessly adapting, in a fully autonomic way, to different operational conditions to fulfill the user requirements. Autonomous actions need to be performed by enabled devices, sensors and/or services with different levels of capabilities. The Pervasive Parking System needs to access and control services that use different technologies in the communication which make interactions and integrations troublesome. This problem can be addressed by employing a common architectural style for implementing the involved interfaces. We propose to adopt the REST approach because interoperability is fostered by the use of standard technologies, the stateless RESTful interactions support scalability, and hypermedia controls reduce coupling between components by driving clients' interaction. Moreover, REST principles provide the opportunity to reuse and generalize the component interfaces, reduce interaction latency, enforce security, and encapsulate legacy systems by using intermediary components.

3 Related Work

In the domain of SOAP/WSDL services, messages are exchanged between endpoints of published applications by using the Web as a universal transport medium. In this way, the applications interact through the Web but remain "outside" of the Web. In addition, SOAP is the single standardized message format in this approach and messages are exchanged in both directions by using only one HTTP verb (POST). In the literature, there are several papers that compare the SOAP and the REST approaches (e.g., [12]). The major advantage of adopting full semantics of HTTP verbs to expose operations is that applications become part of the Web, making it a universal medium for publishing globally accessible information.

In the domain of sensors, the most relevant works are related to OGC standards. In [8], the authors have given a Linked Data model for sensor data and have implemented a RESTful proxy for SOS in order to improve integration and inter-linkage of observation data for the Digital Earth. We have been inspired from their work regarding the URI scheme for observations data, and the creation of meaningful links between data sets. We do not directly address the publication of sensor data as Linked Data but, according to the view presented in [17], we expose sensor data properly linked following the REST architectural style. Having sensor data properly structured on the Web is fundamental whenever integration needs to take place as in sensor networks. Our proposal extends the work in [8] towards hypermedia-driven interactions with OGC compliant services since we consider the explicit definition of the HTTP idioms used for operations, and the processing of observation groups.

Further approaches have addressed the discovery, selection and use of sensors as a service. An advanced approach in [6] offers search mechanisms of sensors that exploit semantic relationships and harvest sensor metadata into existing catalogues. In

[9], a model-based, service-oriented architecture has been used for composing complex services from elementary ones, on sensor nodes. The selection process evaluates the processing and communication cost of the service. In [1], the authors focus on service composition in pervasive systems. They propose ranking services based on context-related criteria so that the selection is based on the service matching score with the composition features. All the above mechanisms need to address critical aspects like the heterogeneity of interfaces and data models mismatches, and thus, can be used in conjunction to our proposal for the integration of sensor data in sensor networks. In [11], the authors describe the selection of services that match user preferences by collecting and evaluating services' descriptions. RESTful interactions can be integrated into such mechanism to facilitate the descriptions discovery and the services selection, and to enable pervasive systems make use of the selection process. Such solution minimizes the number of services and avoids unsuitable services in pervasive systems since it involves only the services that meet the user requirements.

In [4], an approach similar to ours investigates the use of the REST architectural style for providing the functionality of sensors in pervasive systems. It emphasizes the abstraction of data and services as resources, services interoperation via self-describing data and services orchestration with loosely typed components. We extend the authors effort towards a more structural approach of defining the URI scheme for resources, the HTTP idioms for sensors interactions and the application protocol driven by hypermedia links through standard formats. In [14], the DIGIHOME platform has been developed to deal with the heterogeneity, mobility and adaptation issues in smart homes where devices have advanced computational capabilities to improve the user satisfaction, but the heterogeneity of protocols used constrains the integration of these devices into a larger monitoring system. The platform provides software connectors for devices accessed by a variant of protocols such as ZigBee, SOAP and CAN, while HTTP is the communication protocol for the detection of adaptation situations and the handling of events. In [6], the authors explore REST as a mean to build a "universal" API for web-enabled smart things. They give emphasis on the decoupling of services from their representation and the negotiation mechanisms for the representation format, and they propose AtomPub to enable push interactions with sensors, and gateways that abstract communication with non Web-enabled devices behind a RESTful API. Although [14][6] use HTTP according to the REST principles, they do not make explicit how services with conventional interfaces are mapped to a RESTful API. We complement their research by defining rules for the URL writing, and we explore hypermedia controls for hypermedia driven interactions with the service.

4 Design of a RESTful Interface for OGC Services

In this section, we describe how to design RESTful services that are compliant with the OGC's SOS and SPS standards. We focus on activities that allow data sensor requestors to submit sensor tasks and retrieve the generated observations. These are the SPS operations *GetCapabilities*, *DescribeTasking*, *DescribeResultAccess*, *Submit,*

Update, Cancel and *GetStatus*, and the SOS operations *GetCapabilities*, *DescribeSensor* and *GetObservations*. The proposed paradigm enables computer-to-computer interactions since, in the context of sensor data it is a common need to have automated processes elaborating even raw data to proprietary formats.

The overall goal requires first to identify the structural elements of a RESTful interface because these will become the building blocks for RESTful services. The retrieval and managing of sensor data as resource representations requires the identification of (i) the resources of interest, and (ii) the permitted hypermedia-driven interactions of the consumer with the service offering the sensor data. We explore these two dimensions in terms of the URI scheme used to describe resources, the HTTP idioms (methods, headers, status codes) used to interact with the service, the domain application protocol, and the media type for hypermedia-driven interactions.

4.1 Sensor Data as Resources

In SOS and SPS services, resources result from the computations applied to data describing capabilities of services exposing sensor data, sensors as well as their tasks and their observations. The main data classes of objects retrieved are the following: *SOS Capabilities* provides metadata about a SOS service in terms of (i) parameters that may be used to filter the retrieval of observations, (ii) observation offerings used to further organize observations into sets, and (iii) set of sensors associated with the service; *SPS Capabilities* provides metadata about an SPS service in terms of sensors and phenomena that can be tasked; *Sensor* provides the highest level of detail of sensor metadata; *Task* provides information about the status of the tasking request and the task itself; *Observations* groups a set of observations retrieved using the same criteria; *Observation* associates a retrieved value with the sensor providing it, the offering it belongs to, the time period in which it was taken, etc.

The above classes are not enough to describe all the data objects involved in the interactions with SOS and SPS services, and thus, we introduce: the *FeatureOfInterest* to describe the observed stations; the *Observation Offering* to organize observations; the *ObservedProperty* to describe the phenomenon, i.e. temperature, we want to observe; the *Time* to indicate the time period of observations; and the *Procedure* to describe the method used to observe a phenomenon.

URI Scheme. We assign URIs to the objects of the above classes using the conventions proposed in [13] and used in [8, 16]. The base URI for every class has the form: http://my.host/class. Therefore, the URI http://my.host/observations represents a collection of all the observations published by the service. Further segments following the base URI refer to additional criteria to be used for the retrieval and correspond to parameters to be applied to the GetObservations SOS operation. We have identified the following situations in the parameters:

- If the operation call requires *multiple parameters*, then two segments are appended to the URI for each parameter: the first segment indicates the data class corresponding to the parameter name, while the second one indicates the parameter value. Although, this is a strict order for the segments of one

parameter, the segments of distinct parameters can be appended to the URI in an arbitrary way. The retrieved observations are those that satisfy all the criteria indicated in the URI. The *http://my.host/observations/sensors/urn:ogc:object:Sensor:MyOrg:13/observedProperty/urn:ogc:def:property:MyOrg:AggregateChemicalPresence* points to the collection of observations about *urn:ogc:def:propertyMyOrg:AggregateChemicalPresence* taken by the sensor *urn:ogc:object:Sensor:MyOrg:13*.

- If a parameter has *multiple values*, then, since order is not relevant, the semicolon is used to separate the values in the corresponding segment. The retrieved observations are those that satisfy the indicated criteria for at least one of the values. The *http://my.host/observations/sensors/urn:ogc:object:Sensor:MyOrg:13;urn:ogc:object:Sensor:MyOrg:12/observedProperty/urn:ogc:def:property:MyOrg:AggregateChemicalPresence* points to the collection of observations from the *urn:ogc:object:Sensor:MyOrg:13* or *urn:ogc:object:Sensor:Org:12 s*ensors.
- If a parameter has *a range of values*, then, since the order is relevant, the comma is used to separate the start value of the range from the end value in the corresponding segment. The *http://my.host/observations/sensors/urn:ogc:object:Sensor:MyOrg:13/eventTime/2011-04-14T17,2012-04-14T17,tm:rel:between/* points to the collection of observations taken from April 14th 2011 at 5pm and April 14th 2012 at 5pm. The time strings are encoded according to ISO 8601.
- If a parameter has *structured values*, then, since the order indicates the meaning of the value and thus, it is relevant, the comma is used to separate the values in the corresponding segment.

The above rules define the meaning of the URI segments and they are applied to all classes for assembling the URIs of the data objects defined by the class. The first segment after the service host indicates the class whose objects are to be retrieved. The successive segments indicate filtering conditions to be applied to the underlying retrieval mechanism. Because of the different computations that may be applied to retrieve the same resource, we have different URIs representing the same resource. This facilitates the resource access, but it could also be troublesome, if clients are not able to distinguish whether different URIs refer to the same resource. We follow the convention stated above and for each resource we keep an URI that serves as a reference for all the other URIs. Then, we explore hypermedia mechanisms to insert this reference URI in the response every time the client requests other URIs.

4.2 Hypermedia-Driven Interactions with Sensor Services

SPS and SOS services are complementary: the former schedules tasks for collecting sensor data, while the latter publishes the collected data. In particular:

- tasking requests are created when the user makes a submission;
- tasks are created by the service when a tasking request has been accepted;
- a user may retrieve tasks as well as tasking requests, to know their status;
- while a task is in execution, it can be updated and/or cancelled;
- during the execution of a task, observations are created and published;
- observations may be retrieved at any time after their publication.

HTTP Idioms. We distinguish the HTTP methods that may be used for the invocation of the above actions. Tasking requests are created with POST, retrieval of resource representations is achieved with GET, and tasks are updated with PATCH and cancelled with DELETE. We use POST instead of PUT for creating tasking requests because the service and not the user will create and associate an URI with the newly created resource. We use PATCH instead of PUT for updating tasks because the service allows incremental and not overall modifications.

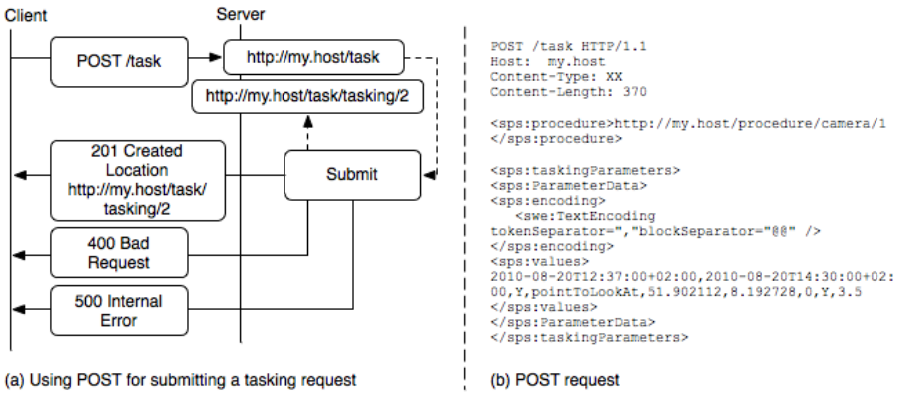

Fig. 2. POSTing a tasking request

In Fig. 2, we exemplify the HTTP request for submitting a tasking request. The request is composed of the POST method, the path that will serve the request at the server and the payload containing the tasking parameters. Upon reception of the request, the control is passed to the Submit operation, but the user remains unaware of this since the operation is inside the server boundaries. The operation creates a tasking request resource and sends the HTTP response to the user. Upon successful creation of the resource, the response status code is 201 Created, and the returned URI identifies its location. In the meanwhile, the system decides whether the requested task will be accepted, rejected or pending. The URI of the tasking request resource identifies also the task to be created and it will be used to GET, PATCH and DELETE the task on the server. Other status codes indicate: (i) a malformed request, and (ii) server failures that prevent the server from fulfilling the request. Similar are the HTTP requests for the remaining actions.

The Domain Application Protocol. Possible interactions a single user may have with both SOS and SPS services constitute an overall protocol for tasking sensors and getting their observations. The partial or complete execution of this protocol changes the state of the resources involved in the communication like tasking request, task and observations. Our method follows the methodology in [16]: resources are the central aspect in the service implementation and the user interactions drive their life cycle.

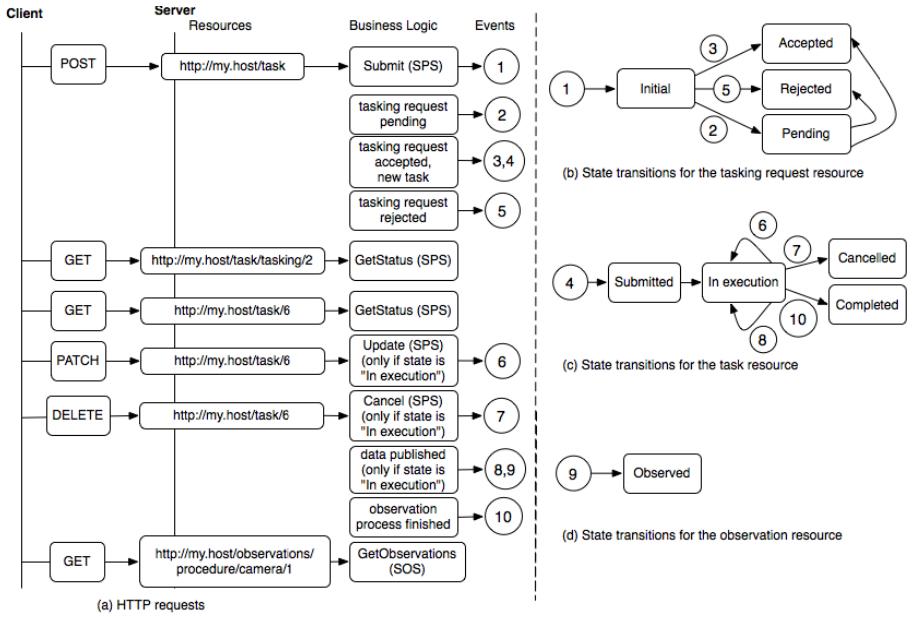

Fig. 3. (a) DAP and (b) resources state transitions

In Fig. 3 (a), we depict the HTTP requests of possible interactions and the events that will fire the state transition for resources. The user POSTs a tasking request, and the SPS Submit operation creates the tasking request resource that enters the *Initial* state (as shown in Fig. 3 (b)). Internal business logic decides whether the tasking resource's state will transition to *Pending*, *Accepted* or *Rejected* state fired by the corresponding events 2, 3 and 5. In case the tasking request is accepted, a new task resource is created, and the event 4 fires the entering of the task resource to the *Submitted* state and then automatically, to *In execution* state (Fig. 3 (c)). After the submission, the user may GET the status of the tasking request and/or task resource. These interactions do not have any server-side effect. As long as the task resource is *In execution* state, the user may PATCH modifications or DELETE the resource firing the events 6 and 7, and the publication of the collected sensor data is enabled. The latter activity fires the newly created observation resource to the *Observed* state (Fig. 3 (b)) and the user may GET its representation. Once the collection process is finished, the task resource transitions to the *Completed* state.

Hypermedia-Aware Media Type. In order to enable the user to create tasking requests and get observations without the need to explore any out-of-band information, we need to provide him with the entry point *http://my.host/task* where he may POST a tasking request, and then insert hypermedia controls in the HTTP responses so that link relations drive the user in the protocol described above. In Fig. 4 (a), we exemplify this mechanism with the HTTP response to the POST tasking request. The response contains both the tasking request resource representation, and links to further steps in the protocol. If the tasking request is pending, then the response provides a link for inquiring again the

current tasking request and the task to be generated (it has the same URI as the tasking request). A subsequent user GET request for the tasking request representation, results to a different response since, in the meanwhile, the tasking request has been accepted and the task is in execution.

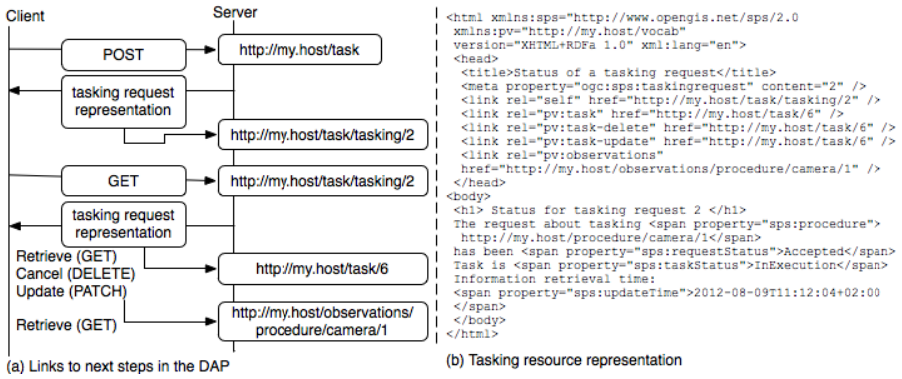

Fig. 4. Hypermedia-aware resource representation

The representation contains further links for retrieving the task resource, requesting modifications and canceling it, as well as links for getting the observations produced so far. The media type for resource representations must understand the semantics of these links in order to enable their automatic interpretation. We have used XHTML+RDFa in order to enable both human-to-computer and computer-to-computer interactions. RDFa extends XHTML with metadata that have the form of triples: subject-predicate-object. Link elements in Fig. 4 (b), convey the location where the task submitted by the current request maybe (i) retrieved, (ii) updated, and (iii) cancelled, and the location where observations may be retrieved. The *meta* element informs that the current context represents an SPS tasking request with id 2. The XML attribute *property* annotates semantically html elements so that a software agent understanding the SPS definitions can extract the enclosed information.

5 Conclusions

In this paper we have made a step towards the development of Self-managing Pervasive Systems by discussing the definition of RESTful interfaces to existing services, and, in particular to sensor services. The adoption of such a widely used communication standard enables for interoperability, which is the basic requirement to give services the capabilities to adapt their behavior. We have shown how Web services can become smarter and deliver more complex functionality by gathering information from sensors and traditional services with minimal human intervention, using inter-linkage of sensor data with hypermedia controls. A first prototype is under development for demonstrating the applicability of our proposal. In the future, we plan to continue in the effort of extending the adoption of REST and Linked Data paradigms to foster the integration of Internet of Things, Internet of Services, and

Internet of People by developing a common and interoperable platform on the existing Web infrastructure. Furthermore, we intend to adopt REST in the contract-driven selection of services [11] in pervasive systems.

Acknowledgements. The work has been partially supported by RegioneLombardia-IBM-UniMiB research grant n.12A135, and project PON01_00861 SMART-Services and Meta-services for SmART Government.

References

1. Bottaro, A., Gérodolle, A., Lalanda, P.: Pervasive Service Composition in the Home Network: Advanced Information Networking and Applications. In: 21st Int. Conf. on Advanced Networking and Applications, AINA 2007, Canada, pp. 596–603 (2007)
2. Botts, M., Percivall, G., Reed, C., Davidson, J.: OGC® Sensor Web Enablement: Overview and High Level Architecture. In: Nittel, S., Labrinidis, A., Stefanidis, A. (eds.) GSN 2006. LNCS, vol. 4540, pp. 175–190. Springer, Heidelberg (2008)
3. Broring, A., Stasch, C., Echterhoff, J.: OGC Sensor Observation Service Interface Standard. Open Geospatial Consortium (2010)
4. Drytkiewicz, W., Radusch, I., Arbanowski, S., Popescu-Zeletin, R.: pREST: a REST-based protocol for pervasive systems. In: IEEE International Conference on Mobile Ad-hoc and Sensor Systems, Fort Lauderdale, Florida, pp. 340–348. IEEE (2004)
5. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
6. Guinard, D., Trifa, V., Wilde, E.: A Resource Oriented Architecture for the Web of Things. In: Proceedings of the International Conference on Internet of Things 2010, IoT (2010)
7. Ibbotson, J., Gibson, C., Wright, J., Waggett, P., Zerfos, P., Szymanski, B.K., Thornley, D.J.: Sensors as a Service Oriented Architecture: Middleware for Sensor Networks. In: 6th Int. Conf. on Intelligent Environment, Kuala Lumpur, Malaysia, pp. 209–214 (2010)
8. Janowicz, K., Broring, A., Stasch, C., Schade, S., Everding, T., Llaves, A.: A RESTful Proxy and Data Model for Linked Sensor Data. International Journal of Digital Earth, 1–22, doi:10.1080/17538947.2011.614698
9. Jirka, S., Bröring, A., Stasch, C.: Discovery Mechanisms for the Sensor Web. Sensors 9(4), 2661–2681 (2009)
10. Palmonari, M., Comerio, M., De Paoli, F.: Effective and Flexible NFP-Based Ranking of Web Services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 546–560. Springer, Heidelberg (2009)
11. Panziera, L., Comerio, M., Palmonari, M., De Paoli, F., Batini, C.: Quality-driven Extraction, Fusion and Matchmaking of Semantic Web API Descriptions. Journal of Web Engineering 11(3), 247–268 (2012)
12. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision. In: 17th International World Wide Web Conference, Beijing, China. ACM Press (2008)
13. Richardson, L., Ruby, S.: RESTful Web Services. O’Reilly, Sebastopol (2007)
14. Romero, D., Hermosillo, G., Taherkordi, A., Nzekwa, R., Rouvoy, R., Eliassen, F.: RESTful Integration of Heterogeneous Devices in Pervasive Environments. In: Eliassen, F., Kapitza, R. (eds.) DAIS 2010. LNCS, vol. 6115, pp. 1–14. Springer, Heidelberg (2010)
15. Simonis, I., Echterhoff, J.: OGC Sensor Planning Service Implementation Standard (2011)
16. Webber, J., Parastatidis, S., Robinson, I.: REST in Practice. O’Reilly, Sebastopol (2010)
17. Wilde, E.: Linked Data and Service Orientation. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 61–76. Springer, Heidelberg (2010)