Title:
Putting Things to REST

Author:
Wilde, Erik, UC Berkeley

Publication Date:
11-16-2007

Series:
Recent Work

Publication Info:
Recent Work, School of Information, UC Berkeley

Permalink:
http://escholarship.org/uc/item/1786t1dm

Keywords:
REST, HTTP, Atom, AtomPub, Web Architecture

Abstract:
Integrating resources into the Web is an important aspect of making them accessible as part of this global information system. The integration of physical things into the Web so far has not been done on a large scale, which makes it harder to realize network effects that could emerge by the combination of today's Web content, and the integration of physical things into the Web. This paper presents a path towards a Web where physical objects are made available through RESTful principles. By using this architectural style for pervasive and ubiquitous computing scenarios, they will scale better, integrate better with other applications, and pave the path towards a "Web of Things" that seamlessly integrates conceptual and physical resources.

Putting Things to REST

Erik Wilde (School of Information, UC Berkeley)

UCB iSchool Report 2007-015
November 2007

Available at `http://dret.net/netdret/publications#wil07n`

Abstract

Integrating resources into the Web is an important aspect of making them accessible as part of this global information system. The integration of physical things into the Web so far has not been done on a large scale, which makes it harder to realize network effects that could emerge by the combination of today's Web content, and the integration of physical things into the Web. This paper presents a path towards a Web where physical objects are made available through RESTful principles. By using this architectural style for pervasive and ubiquitous computing scenarios, they will scale better, integrate better with other applications, and pave the path towards a "Web of Things" that seamlessly integrates conceptual and physical resources.

1 RESTful Things

The fields of pervasive [18] and ubiquitous computing [14] to a large extent have been concerned so far with how to take things which are more embedded into the physical world than the usual computer hardware, and make them available in a way which enables new classes of applications beyond the traditional networking scenarios. We see the "Internet of Things" as a combination of mainly these two fields, where pervasive computing is more concerned with how computing can be better embedded into the everyday world, whereas ubiquitous computing is more concerned with the objects which should be augmented with computing capabilities.

A large class of problems can be subsumed under the heading of *ad-hoc networking* [32], where the nature of the networked things (mobile and not statically configured into an existing networking infrastructure) makes it necessary to think beyond the traditional boundaries of computer networking. The two most important fields in this area probably are *mobile networking* and *spontaneous networking*, both of them tackling the challenge of how to integrate mobile nodes into a networking infrastructure that needs addressing and routing facilities to be able to operate.

Another important area of research has been the question of possible applications in this space, once an ad-hoc networking infrastructure has been put into place. In most cases, the scenarios developed for these applications resemble some kind of information system, with resource types, resource access methods, and resource interaction methods. These information systems often have special characteristics (such as *location* being an important concept and potentially large and diverse populations of resources), but nonetheless are information systems.

Web systems can be designed in a variety of ways. REST focuses on avoiding application state, making sure that important concepts of an application scenario are represented as URI-identified resources, and that all interactions with a client through a server contain all state information that is necessary, so that the

server does not have to maintain session state with clients. This architecture has substantial advantages over state-based applications, which typically work well in their initial phase, but later run into problems with testing, scalability, and integration with other applications.

Figure 1 shows a scenario where a RESTful design of resources allows multiple applications to interact with the same set of resources, without the need to have any API-based interactions between the applications. If the resources are identified and managed in a way which enable the most important interactions with the modeled scenario, then a REST architecture results in a loosely couples system in which resources can be interacted with individually, without the need for one central bottleneck to handle all possible interactions.

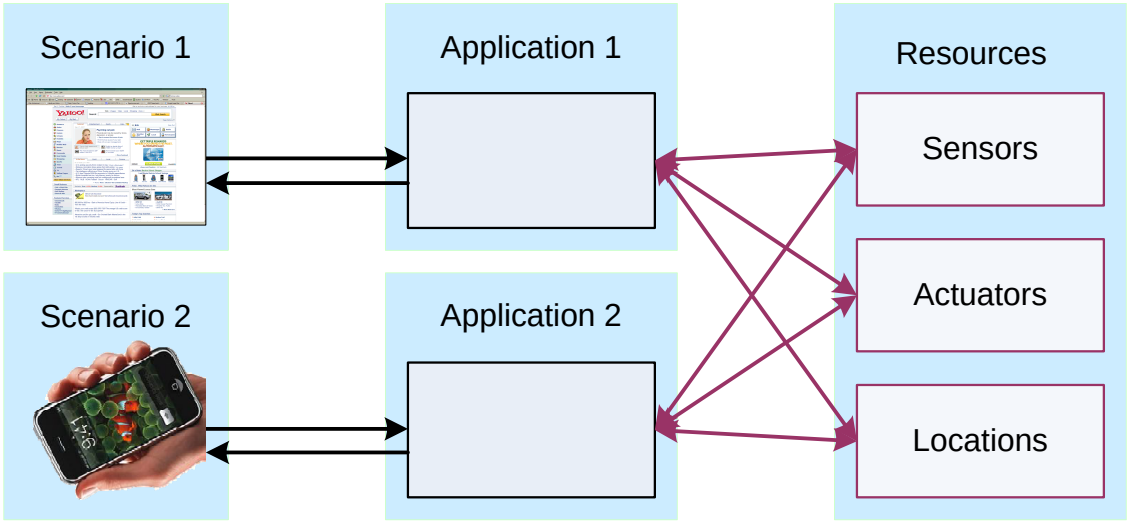

Figure 1: RESTful Access to Resources

Of course, eventually there must be some controlling entity that for example monitors a location, is able to report all the sensors that are active in that location and that can be queried, and handles access to the actuators which can be used in that location. But the REST architectural style allows for the best decoupling of specific application scenarios in this environment, and the basic handling of the resources in it. So while one application might only allow access to sensors for monitoring purposes, another application can also allow access to the actuators for acting in that environment, and both applications can coexist by using the same stateless interaction methods with the resources available.

In this paper, we propose a mapping of these specialized information systems to the information system model of the Web. This paper outlines such a mapping, argues why such a mapping provides real benefits in the short term as well as in the long term (describing two examples and their possible synergies when implementing them in a Web-style way), and identifies areas where the current Web architecture needs to be extended to better support the scenario presented here.

2 Web Architecture

The Web evolved more in a bottom-up manner rather than starting as a top-down designed information system, but there is a design rationale behind many of the Web's technologies, and the most important of these rationales as a whole constitute the *Architecture of the World Wide Web* [23]. While some of the finer points of Web technologies still show their evolutionary beginnings and probably would be designed

differently when starting over today, the fundamental design choices of the Web as a loosely coupled system have been overwhelmingly successful. The most important property of the Web is that it is a *loosely coupled* system which is based on *interchanging resource representations* rather than using APIs.

The most important parts of the Web's architecture are *Uniform Resource Identifier (URI)* [1] for resource identification using an extensible system for identifying resources using a variety of identification schemes, the *Hypertext Transfer Protocol (HTTP)* [11] as the main protocol for interacting with resources in a lightweight and loosely coupled way, and the *Hypertext Markup Language (HTML)* [31] as the primary resource representation which is universally understood and again is a lightweight and loosely coupled language providing hypermedia features. The *Extensible Markup Language (XML)* [3] has become another important resource representation language, it is mainly used for machine-readable resource formats.

HTTP is a rather simple protocol, and the design idea underlying this protocol has been described as *Representational State Transfer (REST)* [12] and has become an important foundation for many Web applications in the last years. The main idea of REST is to design applications which implement their functionality completely as a set of URI-addressable resources, with HTTP being the access method for interacting with them. In such an application, there is no need for any special interface, the application fully blends into the Web and interacting with it does not use any special API, it simply provides access to resources through HTTP.

While the REST architectural style has become more popular, it is not the best style for all applications. However, this paper argues that by using the REST approach to build a "Web of Things", it becomes possible to build a Web which integrates physical resources as seamlessly as possible, and allows new applications using this unified view of the Web of today and tomorrow's "Web of Things". These applications could be considered "physical mashups", where many of the currently popular Web 2.0 technologies to build applications could be used to fundamentally extend the reach of what is possible on the Web.

3 Integration vs. Transport

One of the problems of many Web applications today is that instead of *integrating* into the Web, they mostly use it as a *transport system*. One of the most popular class of examples are the "Web Services" revolving around the *SOAP* [25] and *Web Service Description Language (WSDL)* [5] technologies. The design idea of these technologies is to use HTTP as a transport protocol for API calls, and the API completely hides the resources which are handled by the application. This approach is the exact opposite of Web architecture, where resources (identified by URIs) are the primary means of abstraction (where conceptual work on how to define naming disciplines has started only recently with *URI Templates* [16]), and operations on the resources are modeled as HTTP interactions.

The motivation of using the Web as a transport infrastructure is that the large body of existing approaches of building *middleware* and *distributed systems* can be applied to the Web, by simply mapping these concepts to extensions of the basic API-oriented SOAP/WSDL model. The large number of WS-* additions to the basic model is just a reflection of the increasing functionality of middleware and how all of these functional blocks are being reconstructed on top of SOAP.

The problem with the transport-oriented approach and the already large and continuously growing WS-* protocol stack is that this approach is rooted in the world of building tightly coupled distributed systems. Only peers supporting a complex set of technologies can use the system and the resources managed by it. This approach is appropriate in some scenarios, but we argue that the goals of pervasive and ubiquitous computing should be to provide loose coupling and that there should be a low barrier-to-entry for interacting with resources that are made accessible through the "Internet of Things".

So in the same way as the Web became by far the largest and most successful information system through making its Internet-accessible resources available in a loosely coupled way, the "Internet of Things" should

aim for making its Internet-accessible things available as a part of the Web. This paves the path for the network effect of mashing up the Web and "Things".

Mashups have become one of the big factors of the Web ecosystem, where new applications are generated by recombining existing resources and/or applications. The mashup phenomenon has proven to allow much faster development of applications, because mashups typically can take existing resources and simply put them to use in a new context. While the mashup phenomenon also has shown that some of these applications are not very robust against changes in any of the mashed up components, the possibility to quickly react to information or market needs has become an important factor in developments on the Web.

4 A Web of Things

While the "Internet of Things" establishes the transport capabilities required for networked information systems, we propose to integrate networked "Things" into the Web (as described in Section 3), rather than just making them accessible through non-integrated Web applications. The resulting "Web of Things" is an approach where resources are available through standard Web mechanisms. For example, in a sensor network, each sensor has a URI (it is a resource) and can be queried for its readings (it has state that can be retrieved by accessing the resource).

The big advantage of the integration approach is that "Things" then can be treated like any other Web resource, which means they can be reused in different contexts and applications, and they can be used in a much more open way than just being exposed through APIs which often limit access and interactions to a very specific set of potential users. The approach of fully integrating "Things" into the Web is one step towards the vision of *Physical Computing* [29], where interaction with physical things blends seamlessly into the existing Web of accessible resources.

In the following sections, we briefly discuss examples of how the "Web of Things" could become a very powerful way of integrating physical resources into the Web and employing Web technologies for their access and management.

4.1 Syndicating Things

In many scenarios of pervasive and ubiquitous computing, there is a certain class of objects that are managed in a given environment, and information about these should be made available to a variety of applications. An existing pattern in Web architecture is that of *content syndication*, which traditionally originated in the area of news publishing, but has since been generalized to cover a much broader range of applications.

The *Atom* format [27] is the IETF standard for syndication and is supported by a large variety of existing software development environments. More interestingly, the *Atom Publishing Protocol (AtomPub)* [15] defines a protocol for interacting with Atom feeds. Essentially, AtomPub is a RESTful protocol for interacting with Atom feeds. At the point of writing, it is still an Internet draft, but document editing is finished and the finalized RFC will be published soon. For the approach described in this paper, the interesting part about AtomPub is that it not only defines read interactions with an Atom feed, but also defines create, update, and delete operations, so that it covers the fundamental range of resource interactions.

Figure 2 shows how a typical back-end in a scenario with networked components could be used to drive an intermediate server that allows access to the components through HTTP and AtomPub. While the more traditional HTTP access might only allow read-only access to resources (for example for integrating readings of sensors into Web applications), the AtomPub access could expose more functionality, including manipulation of the resources. This would very likely have to be access-controlled, but AtomPub is based on HTTP and can be easily secured by using HTTP authentication and/or HTTPS.

One possible application scenario for this kind of architecture could be to have a back-end infrastructure with ad-hoc networked nodes which have sensors as well as actuators (Section 5.1 describes a more complex

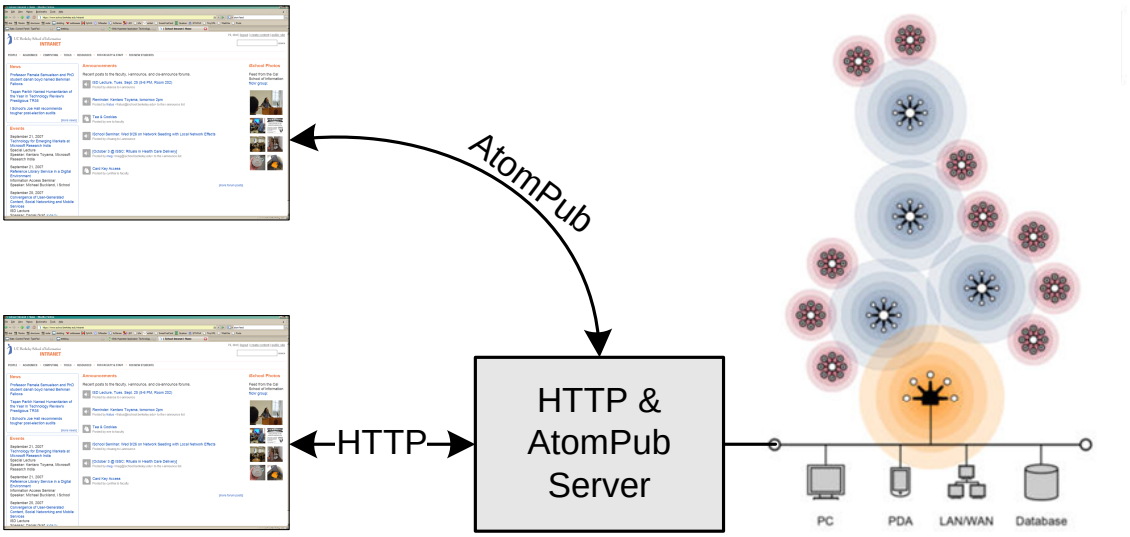

Figure 2: Thing Interaction through HTTP or AtomPub

scenario in the same design space). Simple Web access probably would just request an HTML representation through `GET`ting a Web page, but AtomPub access could provide support for a much more diverse set of interactions (which in REST are often referred to as *basic verbs* because they are the fundamental set of resource interactions that make sense in a large variety of scenarios):

- `PUT`: This verb creates a new resource with a name that is specified by the client. This requires that the naming scheme of resources is well-known by the client, the client has sufficient access rights, and the client supplies a representation of the new resource which is sufficient to instantiate that resource.
- `GET`: Many Web applications support only this verb (with `POST` also frequently being used for form submissions) which returns a representation of the requested resource. It is possible that the representation of the resource is negotiated dynamically in a process called *content negotiation*, which is part of HTTP and allows the client and the server to negotiate the most appropriate resource representation for a given request.
- `POST`: By using `POST`, it is possible to update a resource with new information. An update can either change the state of an existing resource, or it can cause the creation of a new resource, which is then created using a server-generated URI and this URI is returned to the `POST`ing client. These two cases can be distinguished by the status code returned in the HTTP response. In a clean RESTful system, it should be possible to `POST` a new resource representation and then to `GET` the updated resource using the same URI.
- `DELETE`: If a resource is no longer required, the `DELETE` method removes the URI from the accessible resources of a server. This does not necessarily mean that the resource itself will be deleted (or even destroyed, in the case of physical resources), it simply means that the URI that has identified the resource (probably in the context of a given application scenario) will no longer work.

For the resource interactions to be implemented, REST asks for resource representations. For simple HTTP-based requests, this could simply be an HTML page listing the nodes, and updates are often

implemented using HTML forms[1]. For the more advanced AtomPub-based scenario, there has to be a representation which can be used for getting and setting state of the nodes, and this question of resource representations is discussed in Section 4.2.

One disadvantage of the Atom-based syndication scenario is that it follows the pull principle, where subscribers to a syndication feed have to actively pull the feed contents for getting updates. The reason for this is that the Web's architecture is constrained against server-client communications. If this kind of scenario is required (implementing a push or publish/subscribe pattern), then it is likely that the client update part of a system architecture should use other technologies.

Within the context of the *Web Applications 1.0* [19] activities, *Server-sent DOM events* are considered as an extension of the current Web architecture. However, it is unlikely that these will become part of the mainstream Web architecture (because they are violating some other important architectural principles of the Web), and even if they are widely adopted, they will not deliver all the properties on low-latency and high-volume throughput of event streams that some application scenarios might require.

4.2 Representing Things

The REST approach revolves around the notion of *resources* and aims at modeling all client/server interactions as exchanges of resource representations. Interestingly, this leads to a system design without APIs, which is focused on exchanging documents as representations of interactions. This works well on the Web, and it also is getting more popular in commercial data exchange, there the concept of *document engineering* [13] is gaining more traction. This happens because the traditional approach of building tightly coupled systems using APIs is getting increasingly problematic with the number of business partners that typical businesses have, and the need for a robust architecture which can deal with versioning in an environment without one central controlling entity.

The concept of resources on the Web often is confused with the concept of resource representations. A resource is an abstract concept that is assigned a URI and then can be used on the Web. Accessing this URI, however, will never yield the resource itself, but only a representation of it. Resources can have various representations, the most popular ways by which representations can differ are document formats (HTML vs. PDF) and document languages (English vs German). HTTP has built-in features for how clients request certain representations, and one of the powerful aspects of HTTP is this concept of *content negotiation*.

For physical resources (such as RFID tags), it is clear that Web access can never yield the resources themselves. So any resource representation of such a resource will simply be a representation of the current state of the resource. Different applications will be interested in different ways in using this information, so a well-behaving Web server providing access to physical resources should provide various representations. These can then be requested by clients using the HTTP `Accept` header field, so that every client can get the most appropriate resource representation. The following formats are good starting points for making resources usable in a wide variety of use cases:

- HTML [31] is used if a resource's state should be displayed for a human user. HTML is supported on a large range of devices (computers, PDAs, mobile phones) and thus is ideally suited to provide nearly universal access to resource representations. The big disadvantage of HTML is that it lacks semantics and thus is not well suited for representations which should be further processed.

- XML is the de-facto standard for structured information on the Web. Application scenarios can define their own schemas for XML, but in many cases standards or de-facto standards exist and can be reused. Depending on the type of resource, the representation must make the relevant aspects of the resource

[1] HTML forms, however, often do not match well to a real RESTful architecture, because HTML forms only support `GET` and `POST`, and interactions which should be represented as `PUT` and `DELETE` in a clean RESTful design are then also mapped to `GET` and `POST`, making it hard to build a clean URI architecture with an application built around HTML forms.

available (for example, Ye and Chan [37] describe a system where RFID information is made accessible through XML).

- While XML is the most important format for machine-readable data, it is not so well-suited for Web 2.0 applications, where data is read from the server in JavaScript and then used to drive a dynamically updated Web page. Instead of having to parse XML and use it in JavaScript, the *JavaScript Object Notation (JSON)* [6] provides a better solution for JavaScript environments. This representation may be more limited than XML, but can be a good way to make resource data available for Web 2.0 applications.

The above examples focus on the representation of read access of resources, but write access should also be possible, using the `PUT`, `POST`, and `DELETE` HTTP methods described in Section 4.1. For HTML, HTTP methods are limited to `GET` and `POST`, so instead of directly accessing a RESTful service through HTML pages and forms, this requires an intermediate step where HTML's limitations are taken care of and access is mapped to a true REST back-end. XML access can directly use a REST service and is the preferred method of interacting with a resource through HTTP methods. JSON is best limited to read-only access, and any write access to the resource then should also go through the XML representation.

A core question in all these cases is how the representation format is defined. For HTML, this is not really an issue because in most cases this will be a read-only page anyway. However, the XML and JSON formats should be closely aligned. And depending on the resources, some state will be read-only (sensors typically cannot be changed), whereas other state will be read/write (actuators can be read and set). This means that for the write interactions there should be a different representation, one that is limited to the settable properties of a resource.

Ideally, in such a scenario the model of a resource would be defined in a rich modeling language, and then various formats (XML for reading, XML for writing, JSON) could be derived from it. Unfortunately, there still is no established conceptual modeling language for XML [33] which could be used in such a scenario. However, non-XML models such as UML [28] could be used, as long as the limitations of these languages with regard to XML are acceptable.

In a REST architecture, the two most important aspects are the identification and naming scheme for the resources which should be made available, and their representations. Thus, defining the representation of resources should be treated as a core part of making things available on the Web.

The area of how to get from a model to a schema when using XML is still an active field of research, with various competing or complementary schema languages [10, 21, 22], and more advanced approaches looking at a combination of schema languages [36]. Regardless of the approach taken in any scenario, it is important to start from a well-defined model and only then start working on schema(s).

5 Examples

In the following section we discuss two scenarios which can be approached in a way which results in solutions compliant with the architecture of the Web. Thus, implementing these scenarios in this specific way yields benefits as described in Section 4, where it is claimed that one of the most important advantages of a Web-oriented way of building systems is the network effect which can be achieved through adherence to Web architecture. Thus, after looking at the two example scenarios in Sections 5.1 and 5.2, Section 6 analyzes the possible synergies that can be realized when building a Web of things.

5.1 Linkbases

One of the main truisms of today's Web is that there is not a lack of content, but a lack of good ways to use that content easily. Search engines with their sophisticated ranking algorithms have made it possible

to better find resources even when using only simple search terms, but generally, the Web's simple linking mechanism often is not sufficient to properly navigate users through the vast amount of information on the Web. PageRank [30] is an excellent example how an elaborate algorithm has to be used (which is hard to compute, especially given the size of the Web today) to *derive* context from the structure of Web pages and links.

Yet, the Web's simple linking model has also been one of the main contributing factors to its success, only because Web linking is easy to understand in terms of authoring and using it, the Web could become as successful as it is today. When XML was created, for a brief time it was assumed that along with XML there should be a generic language for describing links for XML documents, and the result of this was the *XML Linking Language* [9] defined by the W3C. However, this language is seriously flawed in a number of ways and never got any adoption, so practically speaking, there is no generally accepted linking model for XML.

This is a serious problem, because as described in Section 4.2, one of the main architectural principles of the Web is that of well-defined resource representations. So while it is possible to define structured resource descriptions using XML, it is not possible to define structured relationships between resources using a standardized language.

So while the concept of the *content feed* has gained great popularity, we believe that the concept of a *context feed* will become another important part of the mainstream Web architecture. The idea of a context feed is to provide context for Web resources, so such a feed can be queried with a URI, and returns the context objects that refer to that resource, putting it into context with other resources.

Earlier approaches of using XLink and defining a protocol for context feeds [35] had to start from scratch, because there was no established format for interacting with resource collections. However, the *Atom* format [27] and the *Atom Publishing Protocol (AtomPub)* [15] now provide such a foundation, and context feeds can be easily built on top of that foundation by starting from a representation for context, and a protocol for querying collections of these objects.

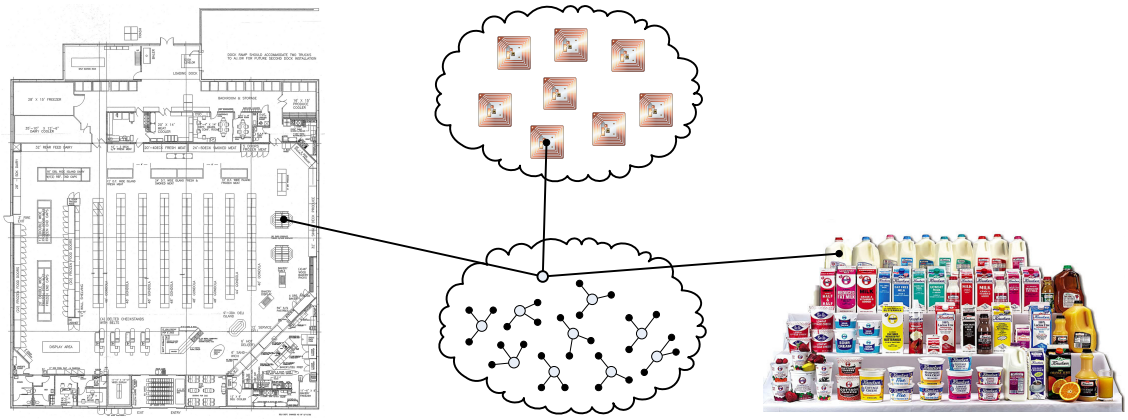

Figure 3: Context as Web Resource

Figure 3 shows an example of such an architecture. The four resource types shown in this figure are RFID tags, products, locations within a store, and context objects establishing connections between the first three resource types. The location concept is discussed in more detail in Section 5.2. The interesting aspect of that architecture is that the first three resource types are independent from each other (unless they are explicitly put into context by context objects), which means they can be independently managed, accessed, and even implemented. If new products are identified, this only affects the product resource type, and if

these products are not RFID tagged, interaction with them will never access any of the other resource types.

RFID tags are managed as individual resources and as soon as a product is tagged with them, this creates context (i.e., a context object is created which represents this connection). For listing an inventory, the flow of controls now would select the product from the product feed, query the context feed with the product URI, and thus retrieve the information about all RFID tags associated with that product. For locative information, the location feed would provide yet another context, as described in the following section.

5.2 Geolocation

Location is a very fundamental property of physical objects. While the Web started as a collection of conceptual resources (mainly documents about research issues), it nowadays contains a large set of resources representing or discussing physical objects. Very different user communities are involved in this, commercial examples are store locations, entertainment examples are movie theaters and tourist locations, and scientific examples are field sciences which discover artifacts for which the location where they have been found is an essential part of the artifacts' description.

Because there are so many communities interested in location information, many different ways of describing and handling location have been invented. The only common denominator seems to be the *World Geodetic System 1984 (WGS84)* [26], the coordinate system that GPS devices use as their internal model. There have been attempts to bring the location concept into the Web architecture [4, 17, 34], but so far all solutions are application specific and there is no generally agreed on location model for the Web.

A true Web geolocation model would have to be consistent among the different parts of the Web architecture. For all parts of the Web architecture there have been individual proposals to standardize geolocation mechanisms, but the proposals for URI [24] and HTTP [8] and HTML [7] lack a common underlying model and thus would not be a good solution, if adopted.

In a scenario such as the one shown in Figure 3, a location concept could be used to communicate about various spatial aspects of the application, for example locating RFID tags in the store and making that information available as another part of an RFID tag's context. There currently is no established location concept for the Web, but the following concept of how to work with location information is well-defined, simple, and can be used in many different scenarios:

- WGS84: The most widely accepted coordinate system for geolocation on the earth is the WGS84 coordinate system. If well-defined and precise location information is required, this system should be used.
- Countries: The only vocabulary of place names that is globally valid is the inventory of countries, which is maintained by ISO [20]. This vocabulary of countries and their subdivisions should be used as an alternative method of location. This is of course is a very coarse grained location concept and does not cover the complete area of the earth.
- Place Names: For more specific applications, we propose to use *place names* which always have to be used in connection with an identifier, which identifies the namespace of the place names. This identifier should be a URI, which in the same spirit as *XML Namespaces* [2] simply serves as a name for a vocabulary of place names. It very likely is not practical to define an authoritative format for place name vocabularies, but specific applications should use formats which are understood by all involved parties.

Using such an approach, the Web would support a very small standard set of location types, and would be open to be extended to any place name vocabulary that applications might want to use. In the scenario shown in Figure 3, the set of place names might include the different sections of the store as well as the warehouse, so that RFID tags could be tracked to the locations that are relevant for this particular application.

6 Synergies

The scenario described in Sections 5.1 and 5.2 identifies four resource types, RFID tags, products, locations within the store, and context objects establishing connections between the first three resource types. One of the most important benefits of modeling these in the style detailed above is that they are loosely coupled. All of them are identified by URIs, and thus the systems implementing the RFID management, the product database, the spatial modeling of the store, and the context management, can be implemented separately. They can also be replaced, when necessary. All that is necessary is that the new system uses the same naming scheme as the old system, and supports the same resource representations. This is the reason why Section 4.2 points out that resource names and representations are the most important part of a RESTful architecture.

Furthermore, in such an architecture new resources could be introduces as required, if the warehouse is upgraded to a fully automated warehouse with robotized access, then all resources which all relevant in that new context have to be modeled in terms of names and representations, and can then be used together with the existing resources. By `POST`ing an RFID URI to the basket of a robotized warehouse cart, the robot would be instructed to add this item to its cart. It could look up the item's location through the context feed, and could update the location by `POST`ing an updated location to the context feed.

Meanwhile, the location concept would work as before, when inquiring about the RFID tags in a certain location, applications could still query the context feed, and the fact how the location of an RFID tag had been established (by manual scanning or by being transported by a robotized cart) would not change anything about the location model of the application.

Generally speaking, by modeling the complete scenario using loosely coupled abstractions, the complete scenario can be treated as a *system of systems*. This results in greater flexibility, better decoupling of the various parts of the application, and better extensibility.

7 Future Work

The RESTful way of designing systems of things as described in this paper is using principles and technologies which are still active research areas. The following areas can be identified as being relevant for the subject of this paper.

7.1 Conceptual Challenges

Many of the approaches of handling resources on the Web are based on the assumption that resources are conceptual entities, and that representations can then be derived from these entities. In the "Web of Things", there is a large number of physically instantiated Web resources, and the question how these interact with conceptual resources need to be addressed. Generally, physical resources will have more constraints than conceptual resources, they have a location, they cannot be replicated, they cannot be cached, and even though some of these actions are also possible, so far there is only little work in this area.

7.2 REST Patterns

REST is still a new concept and while the architectural idea is well-defined and has gained a lot of popularity, there still is a lack of support for designers asking about how to design RESTful systems. The two most important aspects of REST are identification and representation. For identification, the recent work on *URI Templates* [16] might become an important tool for better defining naming schemes for resources.

In the area of representations, there still is a lack of good tools for defining models, deriving schemas from these models, deriving variations of schemas (such as read-only and read/write variants), and deriv-

ing schemas in different schema languages, based on the properties of the model and the implementation environment of a user of the model.

7.3 Unified Location Concept for the Web

Even though there are many application on the Web which support location concepts in one way or the other, there is no unified way of how location information is handled on the Web. The current state is that there are drafts for location information in the three core parts of Web architecture, URI [24], HTTP [8], and HTML [7], but they are not coordinated and their adoption in their current state would mean that the Web would have an inconsistent location model.

Work on the a "Locative Web" should start with a concept for syntax and semantics for locations on the Web. Such a concept should then be used as the foundation for representing locations using URI, HTTP, and HTML, and in addition it would make sense to add location access to the *Document Object Model (DOM)*, so that scripting applications in location-aware browsers could access the browser's location information using the Web location model.

8 Conclusions

This paper outlines an approach for using architectural principles of the Web, in particular *Representational State Transfer (REST)*, as architectural guidelines in pervasive and ubiquitous computing scenarios. We argue that the ultimate goal of these scenarios, the integration of physical objects with the Web of today, should be done using the loose coupling approach that has worked remarkably well for the Web. Research for many of the areas discussed in this paper is still underway, in part because some of the scenarios that Web researchers so far had in mind were not sufficiently informed by the idea of a "Web of Things". Progress in the areas of representing context and making the Web location-aware will very likely be of great benefit to the "Web of Things", and the main goal of this paper is to point out the opportunities and challenges lying ahead.

References

[1] Tim Berners-Lee, Roy Thomas Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. Internet proposed standard RFC 3986, January 2005.

[2] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xml-names-20060816, August 2006.

[3] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). World Wide Web Consortium, Recommendation REC-xml-20060816, August 2006.

[4] Davide Carboni, Andrea Piras, Stefano Sanna, and Sylvain Giroux. The Web Around The Corner: Augmenting the Browser with GPS. In *Alternate Track Papers & Posters Proceedings of the Thirteenth International World Wide Web Conference*, New York, NY, May 2004. ACM Press.

[5] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language. World Wide Web Consortium, Working Draft WD-wsdl12-20030611, June 2003.

[6] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Internet informational RFC 4627, July 2006.

[7] Andrew Daviel and Felix A. Kägi. Geographic Registration of HTML Documents. Internet Draft draft-daviel-html-geo-tag-08, October 2007.

[8] Andrew Daviel. Geographic Extensions for HTTP Transactions. Internet Draft draft-daviel-http-geo-header-04, July 2003.

[9] Steven J. DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) Version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[10] David C. Fallside and Priscilla Walmsley. XML Schema Part 0: Primer Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-0-20041028, October 2004.

[11] Roy Thomas Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. Internet proposed standard RFC 2616, June 1999.

[12] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[13] Robert J. Glushko and Tim McGrath. *Document Engineering*. The MIT Press, Cambridge, Massachusetts, August 2005.

[14] Adam Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, Indianapolis, Indiana, March 2006.

[15] Joe Gregorio and Bill de Hóra. The Atom Publishing Protocol. Internet proposed standard RFC 5032, October 2007.

[16] Joe Gregorio. URI Template. Internet Draft draft-gregorio-uritemplate-01, July 2007.

[17] Amir Haghighat, Cristina Videira Lopes, Tony Givargis, and Atri Mandal. Location-Aware Web System. In *Proceedings of the Workshop on Building Software for Pervasive Computing at OOPSLA'04*, Vancouver, Canada, October 2004.

[18] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober. *Pervasive Computing: The Mobile World*. Springer-Verlag, Berlin, Germany, 2nd edition, August 2003.

[19] Ian Hickson. Web Applications 1.0. Web Hypertext Application Technology Working Group, Working Draft, September 2007.

[20] International Organization for Standardization. Codes for the Representation of Names of Countries and their Subdivisions. ISO 3166, November 2001.

[21] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 2: Grammar-based Validation — RELAX NG. ISO/IEC 19757-2, November 2003.

[22] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based Validation — Schematron. ISO/IEC 19757-3, April 2006.

[23] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.

[24] Alexander Mayrhofer and Christian Spanring. A Uniform Resource Identifier for Geographic Locations ('geo' URI). Internet Draft draft-mayrhofer-geo-uri-01, August 2007.

[25] Nilo Mitra and Yves Lafon. SOAP Version 1.2 Part 0: Primer (Second Edition). World Wide Web Consortium, Recommendation REC-soap12-part0-20070427, April 2007.

[26] National Imagery and Mapping Agency. Department of Defense World Geodetic System 1984. NIMA TR8350.2, Third Edition, January 2000.

[27] Mark Nottingham and Robert Sayre. The Atom Syndication Format. Internet proposed standard RFC 4287, December 2005.

[28] Object Management Group, Framingham, Massachusetts. *UML 2.0 Superstructure Specification*, October 2004.

[29] Dan O'Sullivan and Tom Igoe. *Physical Computing: Sensing and Controlling the Physical World with Computers*. Thomson Course Technology, Boston, Massachusetts, May 2004.

[30] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1999-0120, Stanford University, November 1999.

[31] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. World Wide Web Consortium, Recommendation REC-html401-19991224, December 1999.

[32] Ram Ramanathan and Jason Redi. A Brief Overview of Ad Hoc Networks: Challenges and Directions. *IEEE Communications Magazine*, 40(5):20–22, May 2002.

[33] Arijit Sengupta and Erik Wilde. The Case for Conceptual Modeling for XML. Technical Report TIK Report 244, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, February 2006.

[34] Rainer Simon and Peter Fröhlich. A Mobile Application Framework for the Geospatial Web. In *Proceedings of the 16th International World Wide Web Conference*, pages 381–390, Banff, Alberta, May 2007. ACM Press.

[35] Erik Wilde. Linkbase Access Protocol Design. In *Poster Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[36] Erik Wilde. Metaschema Layering for XML. In Robert Tolksdorf and Rainer Eckstein, editors, *Proceedings of Berliner XML Tage 2004*, pages 106–120, Berlin, Germany, October 2004.

[37] Lei Ye and Henry C. B. Chan. RFID-Based Logistics Control System for Business-to-Business E-Commerce. In *Proceedings of the 2005 International Conference on Mobile Business*, pages 630–636, Sydney, Australia, July 2005. IEEE Computer Society Press.