HOW TO CREATE EVOLVING INFORMATION MODELS BY A LAYERED INFORMATION ARCHITECTURE

Kary FRÄMLING, Jan HOLMSTRÖM

Helsinki University of Technology
BIT Research Centre
PL 5500, FI-02015 TKK
Finland
E-mail: Kary.Framling@hut.fi, Jan.Holmstrom@hut.fi

Abstract:

Flexibility and adaptability of information systems has become increasingly important especially in Internet-based distributed applications. Class-based methods that are efficient in object-oriented programming tend to become too rigid in Internet environments. A two-layered architecture is here analysed as an alternative approach. The paper attempts to show in what sense Internet and the WWW are two-layered architectures and how the evolving "Internet of Things" can be conceived in order to maintain the flexibility of Internet. The Internet of Things provides new and unforeseen innovations also for how industrial enterprises can co-operate and exchange information. The DIALOG implementation of the Internet of Things presented in the paper has been used in such applications and also follows the principles of the two-layered architecture.

Keywords:

Information modelling, Internet, WWW, Internet of Things, Semantic Web

1. INTRODUCTION

The activity of information modeling provides a bridge between two views – that of the real world and the design of an information system [5]. A major challenge in information modeling is how to keep the models flexible enough to allow for future evolution and innovation of the information systems. Parsons and Wand [5] argue that most existing information modeling methods fail to be flexible enough because they are based on the assumption of inherent classification, i.e. that "things" can be referred to only as instances of classes. As a solution, they propose using a *two-layered architecture* to information modeling. The first layer represents instances with their properties, while the second layer consists of class definitions based on sets of properties defining classes of interest. An important property of the first layer is that all instances are unique and are universally identified. For the second layer there is no such restriction; there may be any number of class definitions that allow instances to be classified according to users, views, purposes etc.

In the ideal case, an information system should be flexible enough to provide a kind of "ecosystem", which needs no central control to expand, evolve and create initially unexpected "life forms". In this paper, we argue that Internet and the WWW is such an ecosystem. We will also attempt to show in what sense the Internet can be considered to be a layered architecture.

The *Internet of Things* has been proposed as an extension to the Internet, where it would be possible to access information about any tangible "thing" over the Internet. In order to make the Internet of Things become a similar ecosystem as Internet and the WWW, it would probably be useful to follow their design principles and learn from how they evolved into the current system [6]. At Helsinki University of Technology, the DIALOG research initiative has been developing information architectures for the Internet of Things. In this paper we will attempt to explain how the

DIALOG system conforms to the two-layered architecture in such a way that it could allow for a similar ecosystem to evolve as the current Internet.

The structure of the paper is as follows. After this introduction, section 2 describes the principles of the two-layered architecture and Design Patterns. Section 3 describes in what sense Internet and the WWW can be considered to be an instance of the two-layered architecture and principles for how the new "Internet of Things" should be conceived in order to preserve the same characteristics. Section 4 explains how the DIALOG system has been implemented and how it relates to the previous sections, followed by conclusions.

2. INFORMATION ARCHITECTURES THAT SUPPORT EVOLUTION AND INNOVATION

In organization theory there is no accepted general theory on how to organize to best support evolution and innovation. However, the pivotal role of information and information modeling is widely recognized. Examples of concepts and architectures that explicitly address the issues of evolution and innovation are the two-layered architecture of [5] and the loosely coupled objects used in the Design Patterns introduced in [4]. We will first focus on the two-layered architecture and point out some similarities with loosely coupled objects at the end of this section.

Parsons and Wand [5] claim that the assumption of inherent classification, which is due to the human cognitive way of processing, represents a major obstacle to evolution and innovation. The assumption means that "things" can be referred to only as instances of classes. In reality, the most suitable classification of an instance depends on the context, e.g. the person doing the classification, the organization that handles the "thing" or the purpose the "thing" is currently being used for. To overcome this limitation, they propose adopting a two-layered architecture to information modeling that is consistent with ontological and cognitive principles. The first layer represents unique instances (or "things") with properties, independent of any classes to which the instances may belong. The operations with respect to instances can be generally categorized as retrieval (query) and modification (update) operations. The second layer consists of class definitions based on sets of properties defining classes of interest. The class definitions in the second layer represent different views of the instances that may be created, removed or modified at any time without affecting the instances in the first layer (Figure 1).

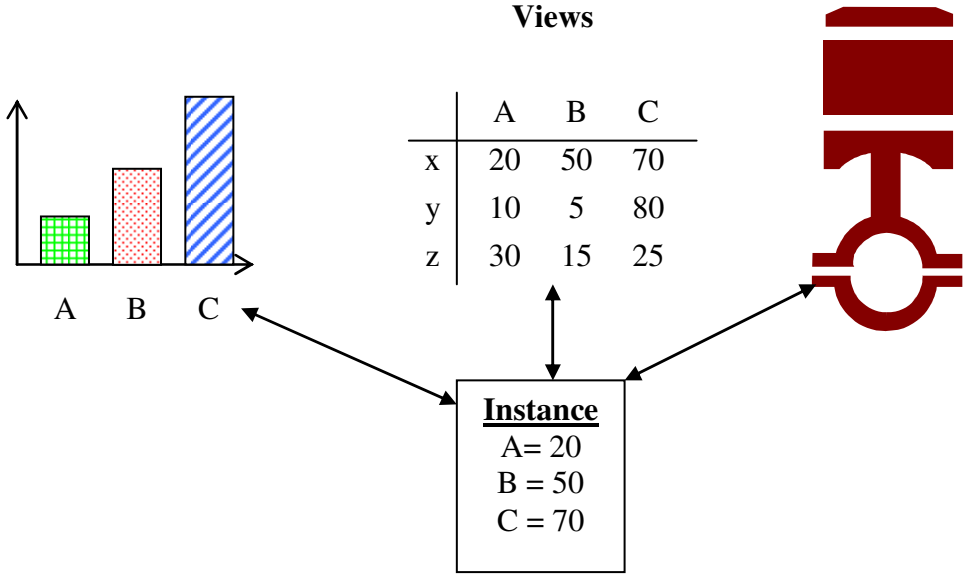

Figure 1: Example of two-layered architecture, where there is one unique instance but several possible views on the instance.

A similar approach can be identified for Design Patterns, where objects correspond to instances (see e.g. [4], p. 5). A general principle of Design Patterns is to use interfaces for defining the set of possible operations rather than using class definitions, thereby enabling so-called *loosely coupled objects*. In practice it means that different objects can have different views of the object that they

have a reference to and that the reference remains valid even though the implementation details of the referenced object would change through time. Object references can also be modified during the whole lifetime of the information system, while class inheritance can only be modified before compiling the whole program. When developing distributed applications that need to run on different computers and possibly communicate over organizational boundaries, it becomes crucial to allow for such flexibility. For a truly distributed application such as the WWW, all modifications have to be done in such a way that the whole system can keep on running. It is unimaginable to stop Internet and the WWW just for adding a new property or functionality and then recompile and restart it all over again (unlike what is done by some operating systems on personal computers).

3. INTERNET AS A TWO-LAYERED ARCHITECTURE – AND HOW TO EXTEND IT FOR REPRESENTING "THINGS"

Internet and the WWW can be considered an ecosystem; it is based on a simple and evolving core of technologies that have enabled it to expand and change in ways that were not expected by the designers [6]. It consists of unique addresses for web pages and a format for presenting and linking the web-pages. More complex functions, such as browsing, viewing, classifying and organizing the information on the web pages is outside the standardized core. This functionality is provided by applications, such as browsers and search engines. In the WWW, web pages are instances with properties defined by HTML-tagged sections, which correspond to the first layer of the two-layered architecture. The second layer is currently provided e.g. by browsers and search engines. Especially search engines try to extract as much information as possible from the available HTML data in order to provide different views of it.

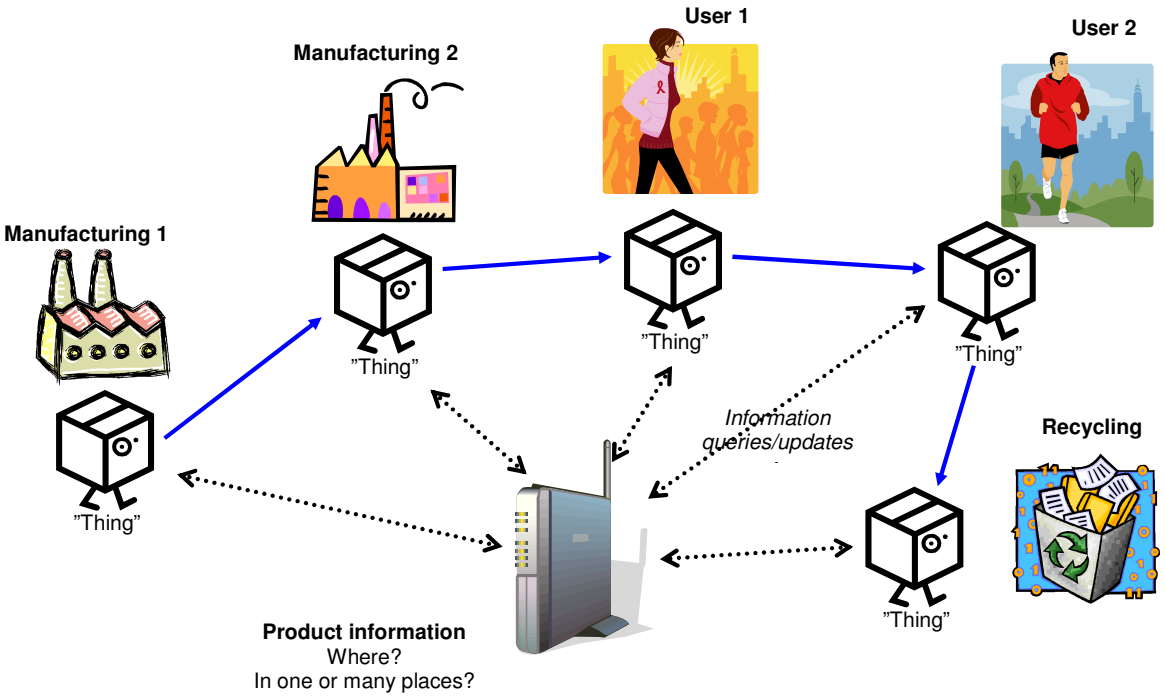

Figure 2: Internet of Things. The „thing" is the unique instance with its properties, while the different users of that „thing" have different views and interfaces to it.

HTML and links between HTML documents allow for loosely coupled references between documents but at the expense of usability. This lack of semantic expressiveness of HTML has lead to various approaches for improving it, notably the Semantic Web (http://www.w3.org/2001/sw/) [1]. The Semantic Web attempts to define common formats and languages for interchanging data and recording how the data relates to real world objects, whereas the original WWW is mainly suited for the interchange of documents. Service-Oriented Architectures (SOA, e.g. Web Services) are technologies that use XML for defining similar interfaces (or views) to data on the Internet as is

done in OOP. Such SOAs are therefore an interesting approach for implementing a two-layered architecture on top of the WWW.

Another extension to the WWW and Internet is the “Internet of Things”. One of the main challenges with the Internet of Things is how to access the information that is not stored locally in the thing itself but is available over the Internet (Figure 2). In 2001, an ID@URI notation and the associated DIALOG information system (http://dialog.hut.fi) were developed at Helsinki University of Technology. DIALOG made it possible to query and update product information about tangible things over the Internet. DIALOG implements the first layer of a two-layered architecture and was used in two multi-organizational pilot installations for shipment and product tracking and tracing. Alternatives to ID@URI and DIALOG are the EPCglobal initiative and the World-Wide Article Information (WWAI) protocol. What is common to all these approaches is that they use “globally unique product identifiers” [3], which is a much broader concept than e.g. record identifiers in databases or object identifiers in object-oriented programs, which only need to be unique within their respective information system or running environment. In the next section we will present how the DIALOG system has been designed for implementing the Internet of Things as a two-layered architecture.

4. DIALOG IMPLEMENTATION OF THE INTERNET OF THINGS

From the communication point of view, DIALOG is a peer-to-peer (P2P) architecture. As for other P2P systems, all DIALOG nodes are equal, i.e. they can both send and receive messages. Therefore any node in the network can ask for information and provide information for the items that it has information about (Figure 3). Other major design principles adopted from P2P are low installation overhead, equality between parties and scalability.

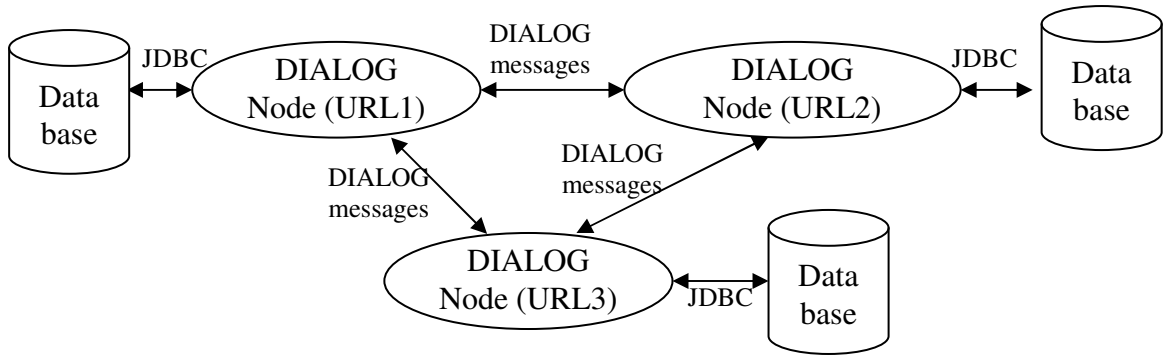

Figure 3. Illustration of peer-to-peer type messaging in DIALOG. JDBC: Java Database Connectivity.

4.1. DIALOG implementation of two-layered architecture

DIALOG nodes by default provide basic instance-level functionality for querying and modifying information about “things”. The “thing” is globally and uniquely identified by the URL address of the DIALOG node (resolved from ID@URI or some other mechanism as explained in section 3) and the ID part that serves as an index for retrieving or modifying data in the database.

The default database contains a table with one record per “thing” with a text field for storing its properties. No particular syntax has been defined for this text field so properties can be encoded in different ways depending on the application (XML, HTML, plain text etc). A special case is if the properties text field only contains an HTTP link, which allows for re-directing a query to an existing web page or service. Another database table is used for storing a history of information updates, i.e. modifications to some property. This table has been used e.g. in shipment tracking for storing all location updates of shipments instead of simply modifying their “location” property with the latest known value. All information of the instance-layer is managed and controlled locally, i.e. in the information system identified by the URI part. This represents an analogy with HTML pages – they

are managed by the organization or person that created them but the information can be copied, linked to or assembled in various ways by e.g. search engines.

Second-layer functionality and how it is implemented is a key element that determines whether an Internet of Things ecosystem can develop or not. An attempt to implement the second layer based on Design Patterns was made in 2003 [2]. The first Design Patterns considered were the "Composite" and "Observer" patterns, which were implemented by using Web Service interfaces with the methods recommended in [4]. Unfortunately, it quickly turned out that by defining new service interfaces, i.e. Observer and Composite interfaces, the flexibility to change and evolve was lost because any modification in these interfaces would make it necessary to update all software that uses them. In the current WWW (or actually the HTTP protocol used) all requests are usually sent to port 80 of the web server. Depending on the received data, the appropriate actions are taken. Therefore the DIALOG system was revised in 2005 so that all semantic information was moved from the service interface into the message contents instead. The functionality needed for implementing different Design Patterns was modified by introducing support for general-purpose semantic nets and agent technology.

The second layer is for the moment supported in the database by a "relations" table that can store relationships between things as (ID1@URI, relationship, ID2@URI2) tuples in the same manner as the Resource Description Framework (RDF) does for semantic relations in the Semantic Web. Such relationships can be used as a representation of general-purpose semantic networks, including the data structures needed for the "Observer", "Composite" and other Design Patterns. The corresponding functionality is implemented by agents; for instance in a tracking context with shipping containers (i.e. "composite things"), a "composite" tracking agent can register for receiving all tracking-related messages. When a tracking message is received, the agent looks for "part-of" relations and forwards the tracking messages as appropriate to the component ID@URIs. As shown by this example, we consider that composition is a functionality of the second layer, whereas [5] considers that it is a functionality of the instance layer. We defend putting composition into the second layer by the fact that real-world "things" can be assembled and re-used as components of different compositions during their lifetime, so in this sense composition is one classification or view among others.

4.2. DIALOG architecture as an implementation of the Internet of Things

The main components or objects of a DIALOG node and their functionality are illustrated in Figure 4. Three "receiver" types exist for the moment that support different messaging protocols, i.e. SOAP (SoapReceiveImpl), RMI (RMIReceiveImpl) and HTML <FORM> messaging (DialogHTMLProductAgent) used by standard CGI- or servlet-type web applications. Which messaging protocol is used depends on how the DIALOG node is started. Any number of messaging protocols can be running concurrently as long as they can be separated by port number, path names or other standard URL components. By supporting CGI-type communication (in addition to Web Services), DIALOG nodes can be accessed directly by browsers and linked to from HTML documents.

The "ReceiveImpl" object is created at startup. The only external method of a "ReceiveImpl" object is the "receive" method that is called by the "receiver" object when a message is received. The "receive" method takes a message object of type DialogMessage as its only parameter and returns an object of the same type. If the received message is synchronous (i.e. requires immediate response), then the message is passed to all "agent" objects that have registered for receiving messages indicated by the "type" field of the message. If the received message is asynchronous, it is added to the receive buffer and given to the appropriate agent(s) by a thread. Examples of agents are "tracking" agents for managing location updates, "product information" agents for querying product information and "observer" and "composite" agents that implement the functionality defined for the corresponding Design Pattern. Agents can also be described by the notion of "plug-in", i.e. a software component that can be used for extending the functionality of the core system at any time.

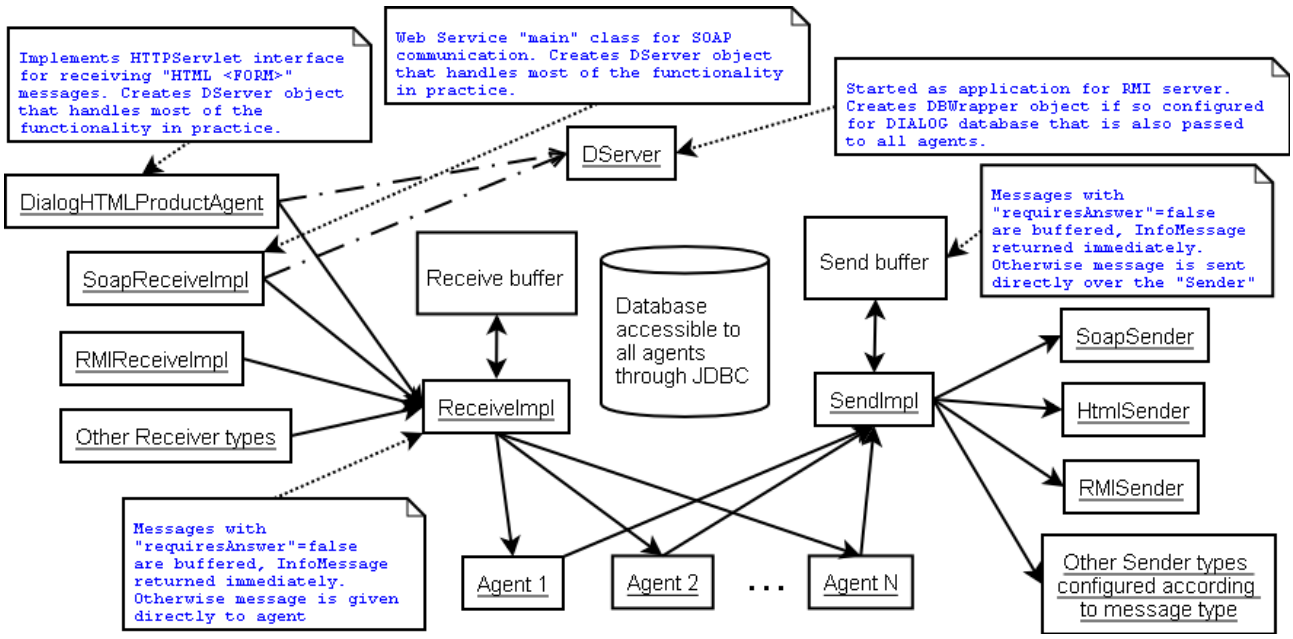

Figure 4. UML object diagram illustrating main components of a DIALOG node and their functionality.

Every agent has a reference to the “SendImpl” object, whose function “send” is called with a DialogMessage object as parameter. The “send” method also returns a DialogMessage object that either contains the requested information (e.g. for a successful synchronous “read” request) or some other status information about the sending of the message. If the message to send is synchronous, it is sent directly to the appropriate “sender” object for delivery and the result is returned if successful. If the message to send is asynchronous, it is buffered and sent by a sending thread using the appropriate “sender” object. The “sender” object to use is determined either from the type of the message (configurable) or the protocol part of the destination URI/URL.

5. CONCLUSIONS

A two-layered architecture provides many advantages compared to “classical” class-based architectures. The WWW and its evolution illustrate how the “ecosystem” provided by such an architecture allows for unexpected applications to evolve. In a similar way, the principles of the two-layered architecture could provide guidelines for implementing the future Internet of Things. If the Internet of Things becomes a similar ecosystem as the WWW, it could also allow for unexpected new solutions in e.g. supply chain management and product lifecycle management, which are domains where information management is currently a great challenge.

6. REFERENCES

[1] Berners-Lee, T. , Hendler, J., Lassila, O., 2001: The semantic web, Scientific American, May 2001.

[2] Främling, K., Kärkkäinen, M., Ala-Risku, T., Holmström, J., 2006: Agent-based Model for Managing Composite Product Information, Computers in Industry , 57/1:72-81.

[3] Främling, K., Harrison, M., Brusey, J., 2006: Globally unique product identifiers - requirements and solutions to product lifecycle management, Proc. of 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM), 17-19 May 2006, Saint-Etienne, France, 855-860.

[4] Gamma, E., Helm, R., Johnson R., and Vlissides, J., 1995: Design Patterns, Addison-Wesley, Reading, MA.

[5] Parsons, J, Wand, Y., 2000: Emancipating Instances from the Tyranny of Classes in Information Modeling, ACM Transactions on Database Systems, 25/2:228-268.

[6] Leiner, B, Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., Postel, J., Roberts, L., Wolf, S., 1997: A Brief History of the Internet, http://arxiv.org/abs/cs.NI/9901011.