DESIGN PATTERNS FOR MANAGING PRODUCT LIFECYCLE INFORMATION

K. Främling, J. Holmström
Helsinki University of Technology, PL5500, FI-02015 TKK, Finland

Abstract
As the number of companies participating in the manufacturing of products increases, the challenges on managing the product life cycle also increase. A major challenge is how to manage product-related information when it is spread on computer systems of multiple companies. It is possible to perform this task in many ways ranging from centralised "portal" systems to distributed peer-to-peer (P2P) architectures. This paper attempts to point out the advantages and drawbacks of these different approaches for managing of product information through the products' whole lifecycle. Design Patterns from object-oriented programming are presented as a potential model for organizing product information and operations performed on it.

Keywords:
Product lifecycle management, Product agent, Peer-to-peer (P2P), Object-oriented programming, Design Pattern

1 INTRODUCTION

The increasing technical sophistication of manufactured products is a challenge for managing their design, manufacturing, maintenance and disposal, i.e. the product's lifecycle. Availability of timely and accurate product information is becoming a necessity during the entire lifecycle. At the same time enterprises become increasingly global and networked (the "virtual enterprise"), which makes it even more difficult to handle product information. A supplementary challenge comes from customized products where every product item has its item-specific information. When considering the entire lifecycle, practically all products are customized products due to different conditions of use, maintenance, spare parts etc.

A product-centric approach has been seen as a solution to handling product information during the product's entire lifecycle [1] [2]. The product-centric approach associates a "virtual counterpart" or product agent with every product item. The connection between the product item and the product agent is maintained by a unique identifier that (directly or indirectly) serves as a reference to the network address of the product agent. A major advantage of the product agent concept is that product information no longer needs to be transmitted and copied between companies that handle the product. As long as the identifier on the product item serves as a reference to the product agent, it is irrelevant where the product agent is hosted. The product agent may also be distributed over several computers.

In this paper, we argue that a reference between the product item and the product agent is not sufficient for developing universal product lifecycle management (PLM) information systems. Many modern products contain an embedded computer that is capable of storing data and processing it autonomously. For such products the entire product agent may be embedded into the product item. In practice, it is application-dependent what parts of the product agent are embedded in the product itself and what parts are located in a backend system. A bottle of mineral water will have no embedded product agent parts, while a car will have embedded product agent parts for on-board control and diagnosis and backend product agent parts at least for design and manufacturing data. A paper factory might embed the entire product agent.

In order to handle all these application scenarios in a uniform way, we need to disconnect the representation of how the product agent is organized from organisational limits and the physical hardware where it resides. In order to achieve this goal we propose using a general object-oriented framework, where product agents can be decomposed into general-purpose objects that are connected by organisation- and hardware-independent object references. In addition to making an abstraction of the underlying hardware, this kind of representation makes it possible to apply well-known methods from object-oriented programming (OOP) such as design patterns and frameworks.

The structure of this paper is as follows: section 2 gives an overview of existing or evolving standards related to product item identification and middleware. Section 3 describes how design patterns from the domain of object-oriented programming could be used for creating adaptable product information systems that are independent of the underlying hardware architecture. Section 4 gives an overview of existing standards and technology for secure middleware communication. Conclusions of the paper are presented in section 5.

2 OBJECT REFERENCES AND MIDDLEWARE FOR HANDLING PRODUCT ITEM INFORMATION

The location of most product items changes at least during some phase of their lifecycle. Therefore they tend to have only intermittent network access (typically through Internet). When they have network access, they may need to communicate with the non-embedded parts of the product agent, for instance for accessing additional information or checking if the backend system has detected a need for maintenance. The minimal requirement for establishing this connection is that the embedded part has to store some kind of reference to the backend system. The communication between the

embedded part and the backend part of the product agent is performed using so-called middleware software that takes care of transmitting messages over the network.

The main requirement for an acceptable reference is that it should be globally unique. In order to be practically usable, the reference should also be easy to transform into a network address without increasing network overhead. The easiest way to accomplish this is to embed the internet address of the backend product agent in the product item itself. In the Dialog approach [3] [4] an ID@URI notation has been used, where the ID part identifies the product item at the URI (Uniform Resource Identifier [5]). The uniqueness of the URI part is guaranteed by the DNS (Domain Name System) infrastructure [6] [7] while the ID part should be unique for that URI. At the minimal level the ID@URI reference can be embedded as a barcode or using a passive RFID (Radio Frequency Identification) tag. In that case the URI should preferably remain the same during the product's entire lifecycle because changing it requires physical access to the product item itself. For more intelligent devices, e.g. smart cards, cars, etc., this should not usually be an issue because they can update the URI themselves if needed. It is also possible to have a list of alternative ID@URI references if uninterrupted access to the backend system is essential. Since the URI part uses existing standards and since there exists many possible standards for the ID part, this approach does not need any new identifier standards. Middleware software was implemented and used in two industrial pilots for tracking shipments in project deliveries [8].

Another approach for creating references between product items and the product agent is the Electronic Product Code (EPC) [9]. An EPC makes it possible to access the URI part through the Object Name Service (ONS) [10]. The EPC is a compact coding that would typically use 96 (or 64) bits for identifying the product item. These bits are converted into an ONS-compatible query that makes it possible to map the EPC into one (or possibly more) URI addresses where information about the product is available. As for the ID@URI notation, the URI part can indicate the format of the information as well as the communication protocol that is used for retrieving it. The main advantages of EPC/ONS are the compact coding of the EPC and the possibility to modify the URI associated with a product item without having physical access to the product item. The major challenge of the EPC/ONS approach is that many of the related standards are still working drafts [11]. Other challenges are that the EPC code still needs to be adopted by commercial actors despite a strong industrial support and that the ONS infrastructure needs to be created in order to be globally usable. Because ONS is an extension of the DNS, companies who want to use the EPC/ONS system will need to register as information providers with an administrating authority.

A different approach is offered by so-called peer-to-peer (P2P) systems that are mainly known for file sharing of music and movies. However, P2P also has many desirable features for identifying nodes in the network as well as individual items. New nodes and items can be dynamically added at any time and are immediately integrated into the network. The network protocol usually takes care of assigning unique identifiers both for nodes and items automatically. Therefore there is no need for an external authority to manage codes as in the EPC approach. Other advantages of P2P solutions is that all nodes can maintain complete control of what data is distributed to whom (even though most file sharing applications do not check or restrict who gets access), good fault-tolerance (breakdown of one node affects the whole network very little) and possibilities to do load-balancing by using nodes that are "close" (in the network communication sense). The World Wide Article Information (WWAI) protocol [12] developed by the company Stockway [13] is partially based on P2P principles [14]. Existing company codes as issued by EAN/UCC or other standardisation bodies identify nodes of the network. When a node has obtained a certificate from a certification authority it can autonomously issue identifiers for individual items (e.g. product items). New nodes are dynamically discovered when appropriate. The WWAI protocol defines messages that enable nodes to exchange any kind of information and link any kinds of objects to each other by named relations (more about named relations in the next section). From a P2P point of view, the main criticism against WWAI is that it requires certificates issued by a certification authority in a similar way as EPC/ONS in order to become an information provider in the network. It seems like this is motivated by the need to find a compromise between existing coding standards and ensuring the uniqueness of the codes.

In addition to these three approaches, emerging web service discovery standards and infrastructure [15] [16] might be a source of entirely different approaches. For instance, a product item could launch a query for its own product agent service at its current location and obtain it dynamically. This is one of the reasons why data structures and algorithms for handling product information should be designed in a way that is not dependent on any particular identification standard or middleware architecture. In the next section the well-known design pattern concept from OOP is presented as a potential solution to reduce the dependency between how product information is represented and the underlying hardware/software platforms.

3 DATA STRUCTURES AND ALGORITHMS FOR PRODUCT LIFECYCLE MANAGEMENT

In software engineering, the organization of data structures and the algorithms that operate on them is a thoroughly studied area. This wealth of existing knowledge and experience should also be used for managing product information. The main challenge in managing product information is that product information is located in computer systems of different companies and organizations, whereas software engineering methods are initially conceived for computer programs running on a single computer (or at least inside a company network).

In this section we will study how standard data structures and algorithms could be used in the context of product information management. Especially the concept of Design Patterns will be studied. A design pattern in OOP is a "well-known" solution model to a given design task that has been tested and documented by experienced programmers [17]. The goals of a design pattern are typically related to reducing redesign and improving adaptability in changing circumstances. Two main principles for improving adaptability are 1) prefer object references rather than hard-coded class structures and 2) program to an interface, not an implementation [17, p. 18]. The reference types presented in the previous section can be used when referring to objects on remote computers, thereby making it easy to turn a local object into a remote one. The same is true for using interfaces instead of direct implementations; if the interface is identical, the only difference between communicating with a local object and a remote object is that middleware software is needed in the case of a remote object.

3.1 Data structures

One of the most basic design patterns is called “Observer”. The intent of the “Observer” design pattern is to define one-to-many dependencies between objects so that when one object (the “Observable”) changes state, all its dependents (the “Observers”) are notified and updated automatically. The “Observable” interface of the pattern defines methods for adding and removing observers while the “Observer” interface defines at least one method for receiving state update messages. One of the most common uses of this design pattern is in graphical user interfaces (GUI), where user actions on one GUI element also affect other GUI elements. Standard GUI classes of the Java programming language are an example of this, where the observer pattern is used in numerous “listener” interfaces.

In the PLM context, tracking of shipments is a typical implementation of the Observer pattern. In shipment tracking, product agents of different companies can express their interest to receive location updates from the observed shipment’s product agent. Figure 1 illustrates the propagation of a LocationUpdate event to two observers using ID@URI references. Product agents at “comp2.com” and “comp3.com” have added themselves as observers for location updates of the shipment ID1@comp1.com (the “Observable” in this case). The identifiers ID2 and ID3 may be the same as ID1 or different. Examples of other PLM application scenarios where the Observer pattern is applicable are for transmitting sensor measurements or breakdown messages of a machine to different product agents. The Observer pattern is in fact a general mechanism for performing synchronized updates of most kinds of information.

The Observer pattern is applicable for communicating state changes of a single product item. In practice, most products are assembled from parts that come from different companies. The different subassemblies typically form a hierarchical structure with “part-of” relations between the subassemblies. This situation corresponds to one of the most important design patterns, the “Composite” pattern. The intent of the Composite pattern is to compose objects into tree structures to represent part-whole hierarchies, where individual objects and compositions can be treated uniformly. One of the most common uses of this design pattern is in drawing programs, where graphical objects may be grouped together to form new objects, which can then be grouped together with others etc. A set of operations is then applicable both to groups and objects, e.g. moving a group of graphical objects in a drawing program also moves all the individual objects. This pattern defines methods for adding and removing “part-of” relations and for navigating through the Composite hierarchy.

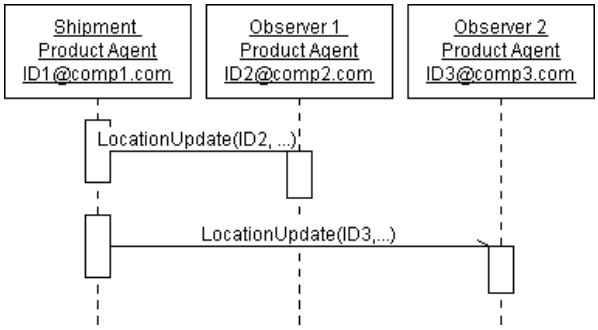

Figure 1: Updating location of a shipment to two Observers in different companies.

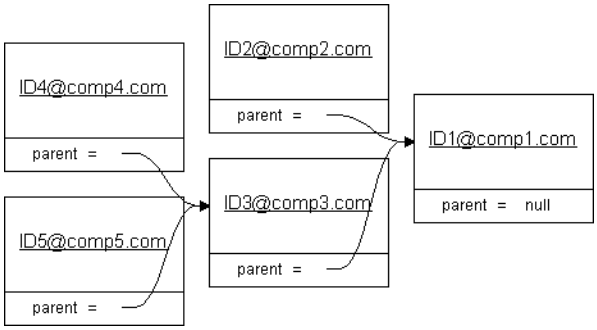

Figure 2: Illustration of Composite hierarchy. Only “parent” references are shown here even though the Composite pattern recommends using bi-directional references, i.e. a list of “children”.

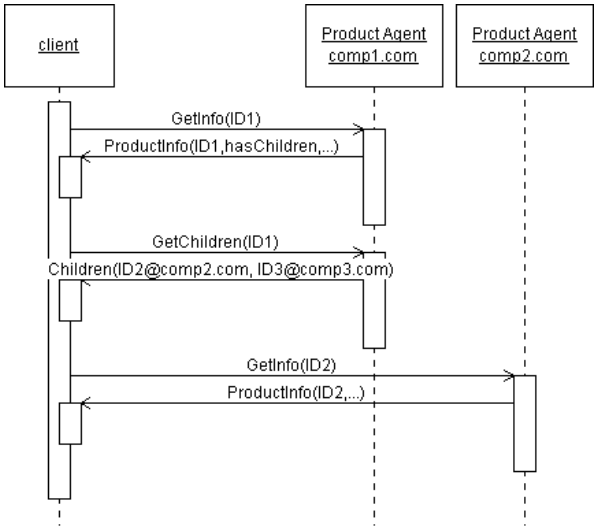

Figure 3: Beginning of sequence for fetching product information for the Composite object in Figure 2.

In the PLM context, information about subassemblies made by different companies need to be linked together. Using object references (e.g. ID@URI) as described in section 2 makes it possible to avoid copying the product information between all the companies in the supply chain. Figure 2 shows an example of a small “Composite” object, where the “parent” references are in ID@URI notation. Figure 3 illustrates how the operation “GetInfo” is performed for the Composite object in Figure 2. The procedure for fetching product information of composite products is an example of an algorithm that can be programmed in many ways. Another algorithm could make it possible to fetch product information from any node in the hierarchy by using “parent” references instead of using lists of children. Structural patterns such as Composite are designed in a way that makes it easy for different algorithms to use them in various ways. Another example of the use of Composite is to define an operation for accessing sensor data or diagnostics information from all subassemblies of the Composite object, independently of the actual type of the subassembly. This is the subject of the next section, where we will use the special case of fault detection and preventive maintenance as an example.

3.2 Algorithms

Algorithms take a set of input data, perform operations with it and produce some end result. In the previous section, the algorithm for fetching product information for composite products used the Composite data structure and the methods defined for it with the goal to get a description of the entire Composite object. There are many well-known

algorithms that perform similar tasks, e.g. Internet search engines that go through the network of HTML links (i.e. references) of web pages on the Internet in order to produce an efficient search database. Another example is different verifier algorithms that are used in CAD systems for construction planning to verify the validity of the design (e.g. resistance, non-collision etc.).

Similar needs also occur in various stages of the product lifecycle. We will here consider the "middle-of-life" case of fault detection and preventive maintenance. We assume that we have a piece of equipment that contains several different subassemblies. A modern car is a good example if such an equipment; it is composed of subassemblies made by a great number of companies. A car also has a rather powerful embedded computer that is capable of some fault detection. Real fault diagnosis still requires using an external diagnosis computer that runs a test algorithm to try to identify the fault and how it could be corrected. The embedded computer has a data structure that gives a partial information model of the parts it is made of and their connections. The diagnosis computer has a similar information model that should be identical to the embedded one for the common parts but more complete. Unfortunately most such information models are constructor-dependent and programmed in an implementation-specific way, i.e. in a way that may differ even between different models made by the same constructor. This means that it is usually possible only for the constructor of the vehicle to program diagnosis tools for them (this may be desired by many constructors but that is not the issue here). If the information model of the car would use patterns such as Composite it would be easier to create generic diagnosis tools.

One of the basic patterns presented in [17] suits this kind of situation: the "Visitor" pattern. The Visitor pattern makes it possible to define new operations without changing the object structure itself. This means that, in the same way as different algorithms can be used for searching the Internet, different algorithms could be used for fault diagnosis of a car without modifying the information model of the car. Visitor also makes it easy to replace only parts of an operation, e.g. replacing the diagnosis part of the fuel injection system without modifying the rest of the diagnosis system. It is easy to imagine other PLM scenarios with similar needs, e.g. accessing dismantling information or identifying all subassemblies made by a given company.

Another example of adaptability and handling changing circumstances is if the embedded car computer is replaced with a much more powerful one, then how could we easily transfer new functionality from the diagnosis computer to the embedded one? And is it possible to use the same data structures and algorithms in a generic way both for embedded systems with very little memory and computing power and for more powerful embedded systems? In this and the previous subsections we suggested using design patterns as a partial solution. The next section attempts to give the rest of the solution by disconnecting the data structures and algorithms from the hardware. This is possible by using universal object references presented in section 2 together with middleware that uses the same design pattern interfaces as the rest of the system.

3.3 Where is the middleware?

A general definition of middleware is that it is software that connects two otherwise separate applications. In this paper we use middleware as a tool to connect software components together in such a way that it becomes more or less transparent whether the components are located on the same physical computer or on another computer.

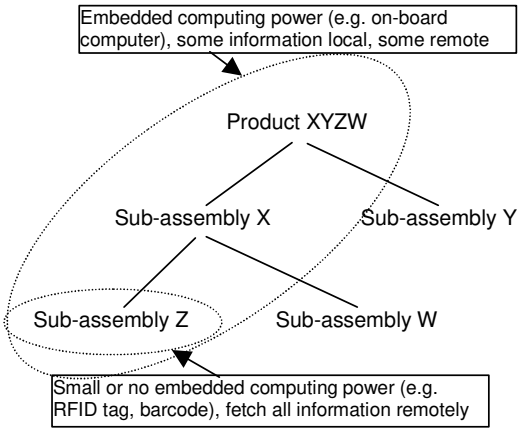

Figure 4: Illustration of "Composite" product items with different embedded computing power. The product structure remains the same but the location of middleware interfaces (indicated by dotted ovals) change.

Figure 4 illustrates why it is important for general-purpose management of product information to make the physical location of software components (and information) as transparent as possible. If there is no embedded computing power in product "XYZW" and subassembly "Z" only has an RFID tag, then a middleware interface has to be used in order to access information about subassembly "Z". On the other hand, if product "XYZW" has sufficient embedded computing power, then it could host its own product agent (or at least parts of it) as well as product agents of subassemblies "X" and "Z" locally, so the middleware interface would rather be at the product level. The point of this illustration is that the product, the product information and the data structure remain the same, only their location changes (i.e. the computer where they reside).

One important difference between local and remote object references is that local references are typically just memory addresses in the same computer, accessed through a program variable. This is not possible for remote object references. Remote object references (e.g. ID@URI, EPC, WWAI, …) need to be persistent even when the software that has created them is not running. Therefore they need to be stored in databases that are managed by the middleware software. Object relations in such as those used in Observer and Composite are characterized by a name (e.g. "part-of", "observes" etc.) and references to the two objects. Such relations can be stored in three fields of a database table as "relation_name;subjectReference:ObjectReference". This is how Observer and Composite relations are implemented in the Dialog platform [18]. The WWAI protocol also supports such named relations.

4 SECURITY

The security considerations associated with product information depend largely on the application area. For instance user instructions of a product may be available without any validation of the product's identity or the identity of the person asking for the information. More restrictive authentication mechanisms are needed when updating product information in the system. Update of the product's maintenance records is an example of a situation where both the identity of the physical item and the person doing the update should be validated.

Several technologies for implementing the desired level of security exist. It is possible to authenticate and verify the

identity of different parties and to encrypt the data being transmitted using standard Secure Sockets Layer (SSL) communication [19]. SSL uses X509 certificates [20] for verifying that all parties are the ones they pretend to be and to initiate a secured communication. The PGP (Pretty Good Privacy [21]) protocol is another alternative. Finally, it is possible to use two-key validation and secure communication directly as explained in [22] (X509 and PGP also use these standards). These are well-known standards that can be implemented by any middleware software.

The main challenge related to security is the management (storage, diffusion) of certificates or keys in a safe way. The WWAI protocol handles certificate management in an elegant way. When a node (a computer) in the network contacts another node for the first time, they both exchange their public keys. From then on they can validate that they are indeed communicating with the same node because only the original node can decrypt messages encrypted with its own public key. When it comes to validating the identity of product items, it becomes difficult for low-range systems such as passive RFID tags. The reason for this is that one encryption key may need to be stored with the physical item itself, which increases the need for storage and processing capacity, therefore also increasing the cost of RFID tags. As a conclusion, security issues can be solved by existing standards but when using low-range systems there is a decision to take between the level of security and the cost for implementing it.

5 CONCLUSIONS

Information systems for product lifecycle management are particularly challenging due to the great number of actors that use or update the product information during the product's lifetime. Focusing product information around the product agent concept instead of trying to push it from one actor to the other is a partial solution. The location of the product agent is irrelevant as long as we have a way of finding a network reference to it. However, the product agent itself is not necessarily located in one single computer system. In the case of composite products the product information may need to be fetched from many product agents that are hosted on different computer systems. Parts of the product agent(s) may also be embedded into the product item(s) itself.

In this paper we have proposed that the representation of product information should be disconnected from organizational limits and the physical hardware. Treating pieces of product information as objects and object references as in object-oriented programming gives access to well known programming concepts called design patterns. Design patterns are generally applicable solutions to many situations. In this paper we have concentrated on the representation of product information for composite products and fault diagnosis algorithms, where the appropriate design patterns offer a good framework for data structures and method interfaces. Finally, we have studied how to disconnect the representation of product information from the physical hardware using these interfaces and various middleware systems.

Even though parts of the concepts presented here have been implemented and tested in an industrial context, most of them still need to be tested and proved operational in practice. What works well in stand-alone programs is not necessarily directly applicable in a multi-company industrial context. Testing this is a major subject of ongoing and future work. Another subject of future work is to identify useful patterns for other product lifecycle management tasks.

6 REFERENCES

[1] Kärkkäinen, M., Holmström, J., Främling, K., Artto, K., 2003, Intelligent products - a step towards a more effective project delivery chain, Computers in Industry, Vol. 50, No. 2, 141-151.

[2] Främling, K., Kärkkäinen, M., Ala-Risku, T., Holmström, J., 2004, Managing Product Information in Supplier Networks by Object Oriented Programming Concepts, Proc. of IMS Int. Forum, Cernobbio, Italy, 17-19 May 2004, 1424-1431.

[3] Främling, K., 2002, Tracking of material flow by an Internet-based product data management system (in Finnish: Tavaravirran seuranta osana Internet-pohjaista tuotetiedon hallintaa), Tieke EDISTY magazine, No. 1, 2002 (Tieke: Finnish Information Society Development Centre, Finland).

[4] Huvio, E., Grönvall, J., Främling, K., 2002, Tracking and tracing parcels using a distributed computing approach, Proc. of 14th Annual Conference for Nordic Researchers in Logistics (NOFOMA'2002), Trondheim, Norway, 12-14 June 2002, 29-43.

[5] Berners-Lee, T., Fielding, R., Irvine, U.C., Masinter, L., 1998, Uniform Resource Identifiers (URI): Generic Syntax, available online (March 5th 2004): "http://www.ietf.org/rfc/rfc2396.txt"

[6] Mockapetris, P., 1987, RFC 1034: Domain Names – Concepts and Facilities, available online (March 22nd 2005): http://www.ietf.org/rfc/rfc1034.txt

[7] Mockapetris, P., 1987, RFC 1035: Domain Names – Implementation and Specification, available online (March 22nd 2005): http://www.ietf.org/rfc/rfc1035.txt

[8] Kärkkäinen, M., Ala-Risku, T., Främling, K, 2004, Efficient Tracking for Short-Term Multi-Company Networks, International Journal of Physical Distribution and Logistics Management, Vol. 34, No. 7, 545-564.

[9] Brock, D.L., 2001, The Electronic Product Code (EPC) - A Naming Scheme for Physical Objects, MIT Auto-ID Center White Paper, available online (December 13th, 2002): http://www.autoidcenter.org/research/MIT-AUTOID-WH-002.pdf

[10] Auto-ID center, 2003, Auto-ID Object Name Service (ONS) 1.0, available online (March 5th 2004): http://www.epcglobalinc.org/standards_technology/Secure/v1.0/WD-ons-1.0-20030930.pdf

[11] EPCglobal Inc., 2005, EPCglobal home pages, available online (March 22nd 2005): http://www.epcglobalinc.org/

[12] WWAI, 2005, World Wide Article Information protocol, available online (March 24th 2005): http://www.wwai.org/

[13] Stockway, 2005, Stockway Oy home pages, available online (March 24th, 2005): http://www.stockway.fi/

[14] RFID Journal, 2003, Peer-to-Peer: RFID's Killer App?, available online (March 29th 2005): http://www.rfidjournal.com/article/articleview/340/1/44

[15] OASIS Open, 2005, UDDI - Universal Description, Discovery and Integration, available online (March 29th 2005): http://www.uddi.org/

[16] OASIS Open, 2005, ebXML - Electronic Business using eXtensible Markup Language, available online (March 29th 2005): http://www.ebxml.org/

[17] Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995, Design Patterns: elements of reusable object-oriented software, Addison-Wesley, Reading, Massachusetts.

[18] Dialog, 2001, Distributed information architectures for collaborative logistics, available online (March 29th 2005): http://dialog.hut.fi/

[19] Netscape, 1996, SSL 3.0 Specification, available online (December 2nd, 2003): http://wp.netscape.com/eng/ssl3/index.html

[20] CCITT, 1988, Recommendation X.509 - The Directory-Authentication framework, available online (March 22nd 2005): http://www.nist.fss.ru/hr/doc/mstd/itu/x509.htm

[21] J. Callas, J., Donnerhacke, L., Finney, H., Thayer, R., 1998, RFC 2440: OpenPGP Message Format, Available online (March 23rd 2005): http://www.ietf.org/rfc/rfc2440.txt

[22] NIST, 2002, Digital Signature Standard (DSS) and Secure Hash Standard (SHS), National Institute of Standards and Technology, available online (December 13th, 2002): http://csrc.nist.gov/cryptval/dss.htm