Software Architecture and Patterns for Electronic Commerce Systems

André Widhani, Stefan Böge, Andreas Bartelt, and Winfried Lamersdorf

University of Hamburg, Department of Computer Science,
Distributed and Information Systems Group (VSIS)

bartelt@informatik.uni-hamburg.de

Abstract

Electronic Commerce is one of the most significant fields in internet applications. With the focus moving from B2C-commerce to B2B-commerce, integrating internet and legacy systems within one company and getting technologies used in different companies to work together is a pretentious task. After discussing the current research in software architecture, including suitable notations for describing them, the aim of this paper is to derive and identify patterns on an architectural level that are specific to the domain of electronic commerce systems. In order to achieve this, we present two case studies featuring selected architectural views on both online shop and electronic procurement systems.

1 Introduction

Creating software systems in the domain of Electronic Commerce with a wide variety of e-business models [Bartelt et al. 2001] is still a challenging task. Although a couple of standard software packages as well as specialized, individual solutions exist, software design know-how in this area is not well documented. Based on current research in software architecture the usage of new patterns described in a suitable notation is proposed. Patterns are derived and identified on an architectural level specific to the domain of electronic commerce systems. In order to achieve this, two case studies featuring selected architectural views on both online shop and electronic procurement systems are presented.

2 Software Architecture

An important role for software architecture is to act as a mediator between the requirement analysis and definition phases and the rather fine-grained system design phase which paves the road to implementation.

As such, a software architecture should not only describe the system in purely technical terms but also capture the language spoken within the domain the system is going to be implemented in. In fact, it should be useful and understood (and understandable) not only by technicians, but virtually any stake-holder involved.

As for many terms, there are a couple of definitions for software architecture. They are not diametrically opposed, but rather emphasize different aspects of software architecture. One of the more frequently used definitions is the following [Bass et al. 1997]:

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.*

One distinguishing feature is a high abstraction level. A software architecture should describe the system on a broad level, before subsequent phases take the design process into detail. This is illustrated by the fact that a component on an architectural level does usually not match a single class or object.

Another vital point to the architecture of a software system is that it has no single face. It is a collection of views on a system that vary both in viewing angle and abstraction level. Commonly used views include functional view, deployment view, business process view and others.

The functional view for example, will describe what functions need to be performed by the system and what domain entities are involved. The functional view can be looked upon from different detail or abstraction levels, and can be focussed on arbitrary subsystems or entities.

Not all views need to be relevant to all stake-holders. The mapping of software components to hardware, which accommodates for availability and fail-over requirements is probably not interesting for domain experts.

Software architecture depicts a software system by describing its constituting components and their interactions. This description is usually performed for different views of the systems.

To be able to effectively create a system's software architecture, suitable tools and a notation is required. During the 1990's, a significant amount of research went into this field. A number of notations where invented. These notations are known under the term architecture description languages (ADL).

As mentioned earlier, different definitions of software architecture highlight different aspects of it. Most ADLs emphasize the composition aspect, with special attention to the connectors which enable component interaction and promote the connector to a first-class entity.

Till now, none of the ADLs have had a major impact on today's software development. One of the reasons accounting to this fact is probably their focus on modeling particular

types of systems. An ADL like Wright [Allen 1997] for example, enables the modeler to perform deadlock analysis for component interaction.

An alternative approach is using the unified modeling language (UML) as a notation for describing software architectures. The advantages are that it is widely known, commonly used and there are a couple of production level tool suites that are mature.

3 Patterns …

Patterns provide solutions for recurring problems, that have proved useful in practice. They add the following benefits to software engineering:

- saving of time and cost because problems are not tackled from scratch
- deeper understanding and better ability of communication amongst stake-holders because patterns often identify and define key terms of the problem
- increased quality and robustness of the actual solution, because it has been tested and refined through repeated use in practice

Patterns like they were introduced by the now famous Gang of Four [Gamma et al. 1994] are largely design patterns. They usually solve a specific design problem within the context of object oriented languages.

[Buschmann et al. 1996] use patterns to describe software architectures while [Fowler 1997] uses patterns to solve recurring problems in specific application domains.

They also introduce a hierarchy of patterns:

Architectural patterns describe a significant part of or view on a system. They sometimes define a certain style that is used throughout. One example is the model-view-controller pattern used in interactive applications.

Design patterns solve problems that are closely related to a specific programming paradigm. A significant part of literature related to patterns today focuses on design patterns, and especially those that are being used in the context of object oriented programming languages, in particular Java. These are patterns like façade, business delegate, bridge pattern and many others.

Idioms are most low level, often fragments of code and applicable only to a certain programming language. An example of an idiom is a reference counter for dynamically created objects in C or C++, in order to support efficient memory management.

We propose another class of patterns, which we call domain patterns.

Unlike architectural patterns, which are rather technical and generic with respect to the area the actual software system is being used in, domain patterns describe organizational structures or business processes that have proved useful within a particular application domain. One example is the order process used in electronic commerce systems.

In the remainder of this paper the focus will be on domain patterns, which are particularly useful in software architecture. They aid in establishing a common language, help identify and document domain specific entities and processes and preserve knowledge gained in subsequent software projects within a specific application domain.

4 … and Domain Architectures

In most cases the application domain for which a new software system is created existed long before - which means that the domain already has a certain set of standards or reference literature and there is a defined vocabulary for entities and processes in that domain.

It should be noted, though, that processes might change when adapted to information systems and in some cases might even make new processes possible that did not exist before, either because they would have been to costly or just not feasible. One example is realtime order and delivery tracking.

Two case studies are presented which examine two vital elements of electronic commerce systems - the catalog and the order process. They are performed for both online shops and electronic procurement systems in order to see where they differ and whether reuse of architecture through patterns is possible with these two related application areas. UML is used as a graphical notation, while the pattern description roughly follows [Meszaros et al. 1996].

The Catalog is a central part of any Electronic Commerce system. There are a number of different ways how to exactly organize products in a catalog. We try to give a blueprint for the organization of a catalog, discuss alternatives and compare it with common structures found in today's commercial products like Intershop Enfinity and standard E-Business formats like BMEcat, xCBL or cXML.

After looking at the catalog, which represents a rather static structure, the look is at a dynamic (business) process which is the order process. The order process involves user interaction with the front-end system as well as other systems in the background.

Existing deficiencies of UML are exposed and it is elaborated on the overall suitability of UML, if applied to modeling on an architectural level.

5 Architectures for Electronic Commerce

Before proceeding to the case studies, short definitions for both online shop and electronic procurement systems are given.

The online shop [Bartelt et al. 1999] is probably the most prevalent type of electronic commerce applications. Regarding involved actors, it is categorized as a B2C (Business-to-Customer) application. The retailer is selling goods to consumers.

Electronic procurement systems enable business between non-private parties. More precisely, a certain set of offerings is gathered from multiple vendors into a single view for the customers. Today electronic procurement application are usually centered around so called MRO goods (Maintenance, Repair and Operation), which are essentially items that are not directly involved into the value added process. Furthermore, electronic procurement applications are divided in two classes: supplier-centric systems and buyer-centric systems [Bruins et al. 2000; Georgantis et al. 2002].

In a supplier-centric system, the supplier manages and updates the catalog. This involves no or little costs for the buying party, but makes the actual procurement activity more complicated as soon as multiple vendors are involved.

In a buyer-centric system, the purchasing party manages the catalog of the procurement system, which might be a part of the corporate intranet. Aggregating the contents of several possibly different catalogs from multiple suppliers is challenging, to say the least. The biggest advantage is the simplification for the purchaser, as he or she does not need to visit multiple websites with different user interfaces. Also, the purchasing party has full control on what catalogs or items are displayed for a particular user.

Another business model is the marketplace. Unlike online shops, where one vendor serves multiple customers and electronic procurement, where one vendor servers one customer at a time, in the marketplace multiple vendors interact with multiple clients.

5.1 Electronic Commerce Catalog Pattern

Context and Forces:

Before attempting to come up with a suitable structure for catalogs, we need to take a look at the requirements. As the catalog is a means to organize products, these requirements are of organizational nature:

- support nested categories
- items should be able to be assigned to multiple categories
- intelligently support variants of a base product (example: one item that is available in different sizes).
- attributes of products should be flexible enough to support each products peculiarities, but should also ease assignment of common attributes for all products or products of a certain type
- compatibility with major classification schemes, like UN/SPSC [Unspsc 2002] or Ecl@ss [Eclass 2002]

Solution:

A simple, straightforward model for the catalog is shown in figure 1.

The catalog serves as a root node. Below this root node we have a number of categories. Categories itself may be nested, the nesting may be of arbitrary depth. At the same time categories may have products assigned to it. Note that there may be categories that have no products assigned to, but a product needs to be linked to at least one category.

This structure fulfils the first two requirements that were proposed. Now we need to take a look at the products attributes which fall into a set of different types of properties, some of which are constrained in a certain way, while others have no such restrictions.

A suggested solution is shown in figure 2. There are properties that each and every product has.

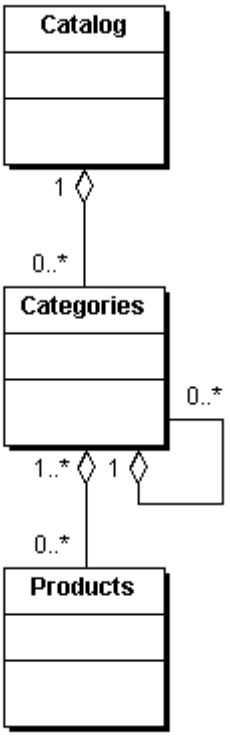

Figure 1: Basic Catalog pattern (UML class diagram)

We call them fixed attributes, and these are properties like manufacturer, identification code and textual description. Product type attributes are those attributes that certain kind of products have in common. An example for a product type attribute is the number of pages for a product type book. Note that we refer to product type, not categories. Categories might or might not group products of the same type under one category. Categories are presentation-oriented and are not necessarily related to product types.

The product type in our catalog pattern maps to the classification scheme used in the mentioned standards mentioned above.

Finally, a base product may have variations of one product, that differ in color or size, for example.

Variations:

Although not stated in the diagram, you would usually assign a price to a product variation, not the base product, because certain variations may be more expensive than others. You may have a default price associated to a base product, though, in order to avoid the need to specify it for each variation in case it is same.

Finally, there may be the need to assign attributes that are freely definable. We call them custom attributes. This caters for information certain manufacturers specify, while others that deliver products of the same type may not be able or willing to provide, or that stem from different depth on information transported by different catalog exchange formats, which particularly applies to electronic procurement applications gathering data from a couple of source catalogs.

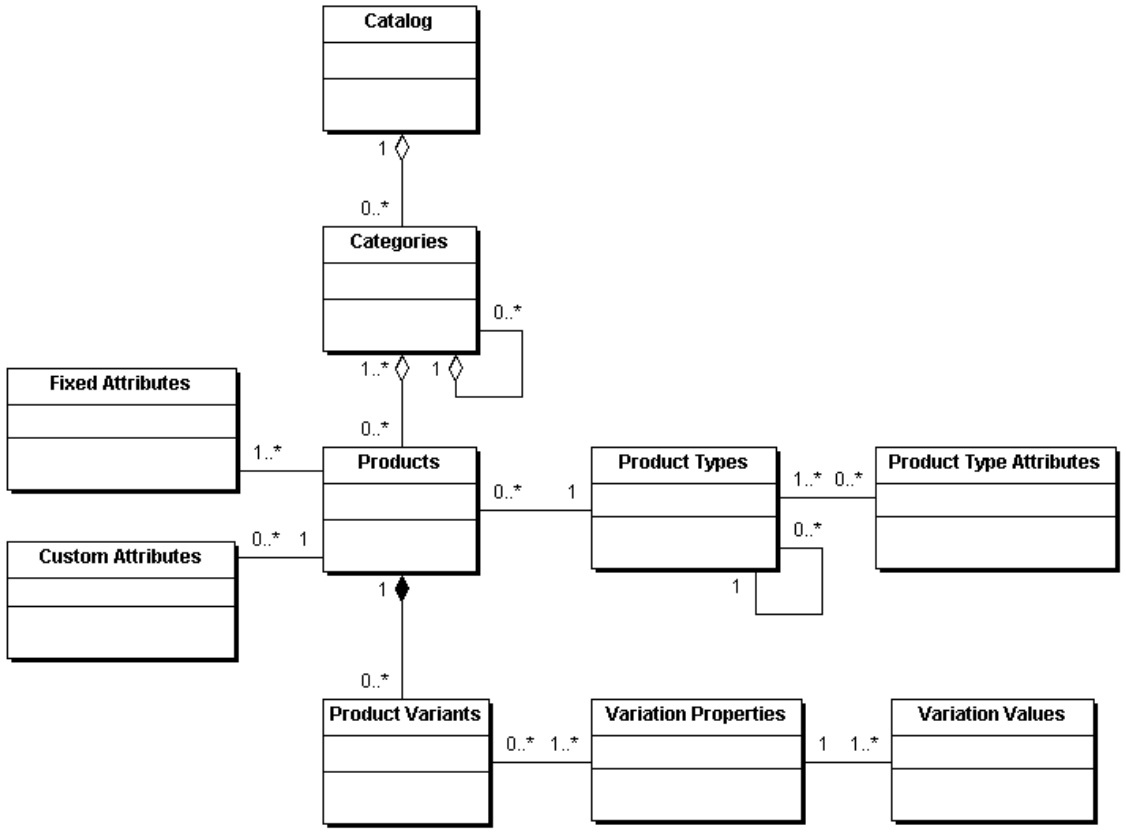

Figure 2: Electronic Commerce Catalog pattern (UML class diagram)

While this catalog pattern enables you to cope with most classification schemes in use today, there are a number of subtle questions that need to be answered during the planning and implementation of such a system. How do you assign products to multiple categories? Does each of the affected categories have a reference to one and the same product or is a new instance of this product being created alongside ? Note that this is not an implementation level decision. It rather depends on whether one has the requirement to assign different attributes depending on where the product is presented.

Currently available standard software packages handle this in different ways. Intershop Enfinity [Intershop 2002] assigns products by reference. Hybris Jakarta [Hybris 2002] allows you to have a copy of one product inherit all properties of the master product, and explicitly lets you overwrite attributes as required.

The catalog pattern may be used equally well for online shop and electronic procurement applications. The main difference does not lie in a suitable static structure, but rather in the process of populating the catalog.

It can also be used with catalog exchange formats like BMEcat [Bmecat 2002]. BMEcat has concepts and tags related to the catalog hierarchy, a so-called feature system that maps to product type attributes and classification schemes and product variants as well as user defined extensions that equal custom attributes in our catalog pattern.

Currently, there exist a number of standards for both classification and identification of products. A classification scheme groups related items into same categories, whereas an identification scheme assign unique codes to products in order to unambiguously identify them. Discussions of various identification and classification schemes can be found in [Beckmann et al. 2001] and [Granada 2001].

In supplier-centric electronic procurement applications, multiple catalogs are merged into one catalog in the procurement system. The use of one and the same classification scheme is essential for easy integration. If the source catalogs use different classification schemes, the process of merging these becomes intricate. Possible solutions include an approach to map one grouping scheme to another, which is beyond the scope of this paper.

Another difficulty lies in the diversity of catalog exchange formats. There are a couple of possible strategies to map one catalog exchange format to another. Almost all of them employ Extensible Stylesheet Language Transformations (XSLT), see [Buxmann et al. 2001; Omelayenko et al.2001a; Omelayenko et al.2001b] for examples.

5.2 Order Process Pattern

Context and Forces:

After having browsed through the products, the customers eventually proceeds to the order process. The requirements are simple:

- personal information of the purchaser, like payment and delivery details need to be collected
- the purchaser should get a confirmation
- order information must be passed to the relevant systems or persons for fulfillment

Solution:

A straightforward order process for online shops is presented below in figure 3.

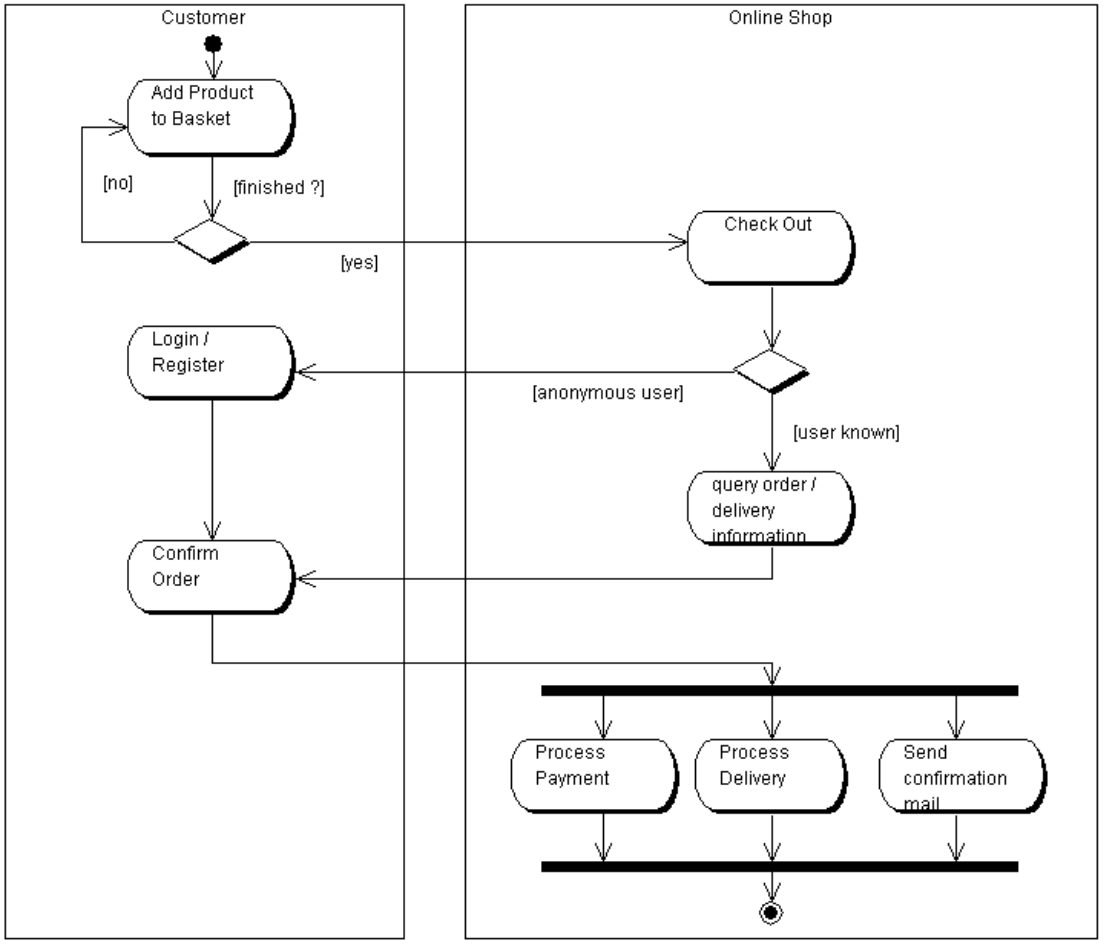

Figure 3: Order process for online shops (UML activity diagram)

The customer adds products to the basket and finally enters the order process. In an online shop, the purchaser may or may not have registered or logged in. Online shops usually defer the process of collecting personal information until absolutely needed in order not to scare off potential buyers from browsing through the offerings.

The payment and delivery processes might be tightly integrated with the shop platform through payment and ERP systems or might be handled manually.

Variations:

An electronic procurement application, particularly one that is buyer-centric, usually is part of a corporate intranet. In this case, proper authentication is handled right from the start in order to guarantee that the person is authorized to use the application.

Another significant difference is that electronic procurement applications need an additional workflow subprocess which is part of the order process itself, as seen in figure 4. During this process, orders can be queued for manual approval and budget constraints can be checked.

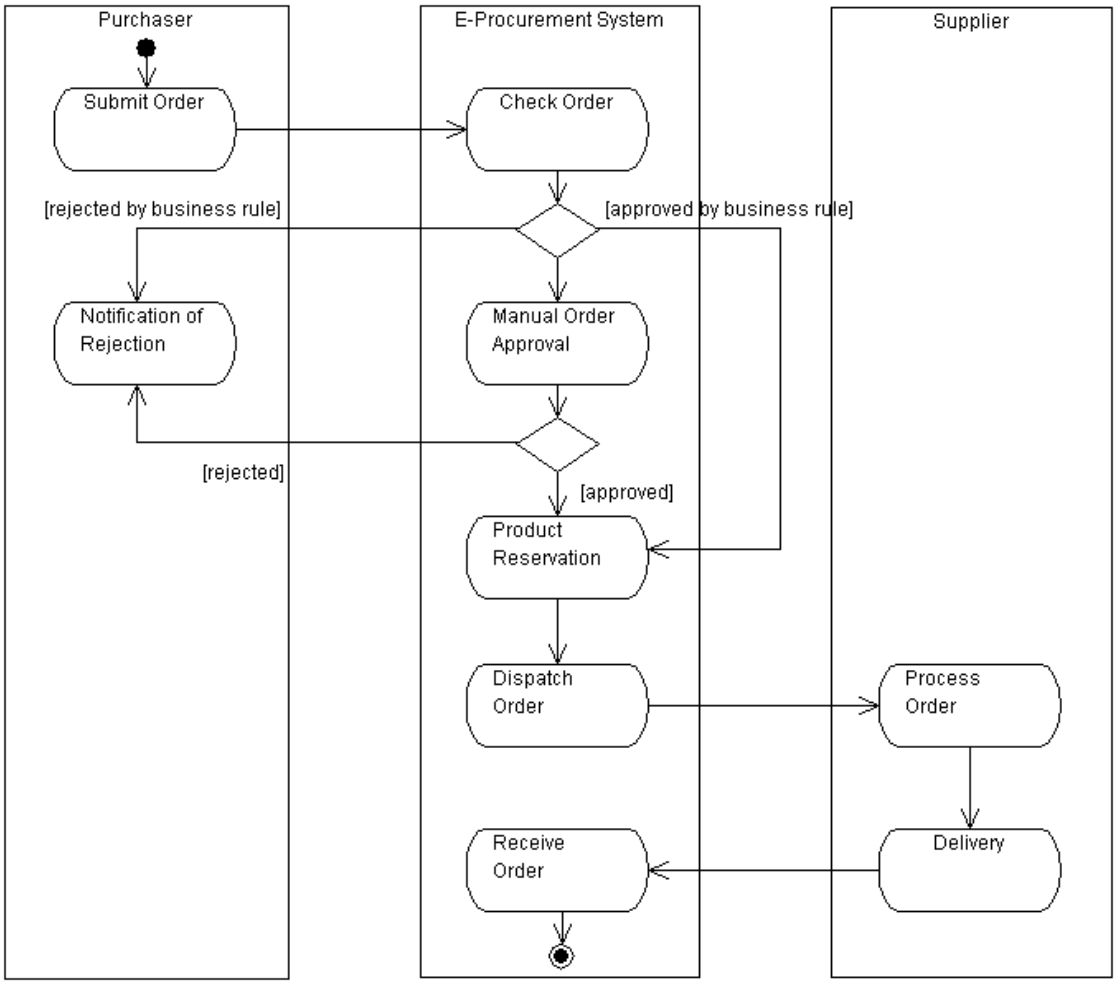

Figure 4: Authorization workflow for electronic procurement (UML activity diagram)

The order comprising of multiple products needs to be split into sets that correspond to one supplier each. Finally, the system must be able to handle invoice and billing functions.

6 Conclusion

The Unified Modeling Language has been used as notation throughout this paper. Using UML for architectural diagrams has drawbacks, though. First, there aren't really rules regarding what type of diagram to use for which kind of view, which stems from the fact that UML has not been designed as an architectural notation in the first place. The decision to choose the appropriate diagram type can be difficult for the person creating these diagrams and confusing for people reading them, because they might, for example, not be acquainted to class diagrams being used to describe domain entities which may or may not have anything to do with classes. Also, different people might use a different type of diagram for one and the same view.

In lack of convincing alternatives, we would still recommend to go with UML. Architectural issues are getting into the focus of UML's driving forces, although proposals are often not very concrete.

Regarding online shops and electronic procurement, it can be said that there exist patterns which capture best practices within these application domains. During the process of finding appropriate solutions it is important not to ignore concepts and reference solutions as well as requirements that have already emerged over the time within the application domain outside the context of information systems. These patterns can and already do serve as building blocks for domain specific standard software packages and frameworks.

The online shop is the predominant business model among electronic commerce applications existing today. Electronic procurement applications are emerging and most vendors that offer standard software packages for online shops are extending them to be used for electronic procurement.

Some requirements, like authorization workflow can be added with comparatively little efforts. Integrating multiple catalogs, which possibly employ different classification or identification schemes into a single application, though, is a major challenge.

References

Allen, Robert (1997): A Formal Approach to Software Architecture, in: Ph.D. Thesis, CMU-CS-97-144, School of Computer Science, Carnegie Mellon University

Bass, Ken; Bass, Len; Clements, Paul; Kazman, Rick (1997): Software Architecture in Practice, Addison-Wesley 1997

Bartelt, Andreas; Meyer, Jochen (1999): A Practical Guideline to the Implementation of Online Shops, in: Bob Werner (Hrsg.): SRDS'99, IEEE Computer Society Press, pp. 348-353

Bartelt, Andreas; Lamersdorf, Winfried (2001): A Multi-Criteria Taxonomy of Business Models in Electronic Commerce, in: L. Fiege, G. Mühl, and U. Wilhelm (Hrsg.): Middleware 2001, WS on Electronic Commerce, Springer-Verlag, Berlin Heidelberg, pp. 193-205

Bmecat (2002): Website of the eBusiness Standardization Committee, http://www.bmecat.org (requested on August 15th, 2002). Some information can only be retrieved after registration.

Bruins, Arnout; Steen, Maarten W.A.(2000): Electronic Procurement

Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael (1996): A System of Patterns, John Wiley & Sons, 1996

Buxmann, Peter; Martin, Luis; Wüstner, Erik (2001): XML-based Supply Chain Management – As SIMPLEX as it is –, Freiberg University of Technology, Chair of Information Management

Eclass (2002) : Website: eCl@ss, http://www.eclass.de (requested on August 15th, 2002)

Fowler, Martin (1997): Analysis Patterns : Reusable Object Models, Addison Wesley, 1997

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1994): Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994

Georgantis, N.P; Koutsomitropoulos, D.A.; Zafiris, P.; Papatheodorou, T. (2002): A Review and Evaluation of Platforms and Tools for Building E-Catalogs

Granada Research (2001): Using the UN/SPSC – Why Coding and Classifying Products is Critical to Success in Electronic Commerce

Hybris (2002): Website: http://www.hybris.com (requested on August 15th, 2002). Some parts of the website are password-protected.

Intershop (2002): Website: www.intershop.com (requested on August 15th, 2002). Some parts of the website are password-protected.

Meszaros, Gerard; Doble, Jim (1996): A Pattern Language for Pattern Writing, PloP ‘96 Proceedings

Omelayenko, Boris; Fensel, Dieter (2001a): A Two-Layered Integration Approach for Product Information in B2B E-commerce, Division of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081a, 1081hv, Amsterdam, The Netherlands

Omelayenko, Boris; Fensel, Dieter (2001b): An Analysis of Integration Problems of XML-Based Catalogs for B2B Electronic Commerce, Division of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081a, 1081hv, Amsterdam, The Netherlands

Otto, Boris; Beckmann, Helmut (2001): Klassifizierung und Austausch von Produktdaten auf elektronischen Marktplätzen

UNSPSC 2002: United Nations Standard Products and Services Code, Website: http://www.un-spsc.net (requested on August 15th, 2002)

