Building E-Commerce Applications from Object-Oriented Conceptual Models

Oscar Pastor*
and
Silvia Abrahão*
and
Joan Fons*

This paper introduces an extension to UML that takes care of web page navigation using the OO-Method, dynamic prototyping, and a new way of specifying the navigation design. Furthermore, a software production process for e-commerce applications design is described. This process is driven by an Object-Oriented Web-Solutions Modeling approach (OOWS), which provides mechanisms to deal with the development of web-based applications. In the proposed process, a system is completely specified using object-oriented conceptual modeling techniques to capture properly the specific functionality of an e-commerce application. We place special emphasis on a navigational model that provides abstraction primitives to capture and represent navigational semantics. With such an extended conceptual model, system functionality and navigational features are described within a unified framework. Once the system specification is completed (problem space level), a strategy to obtain the software components that will constitute the final software product (solution space level) is defined. We briefly discuss how to map these abstraction primitives to e-commerce applications in order to be able to go from the problem space to the solution space in a structured, automated way.

Additional Key Words and Phrases: Conceptual Modeling, Object-Oriented, E-Commerce.

1. INTRODUCTION

E-commerce is fast becoming popular as a way of delivering of business processes and applications over the Web. More and more organizations require the implementation of web-solutions with functionality to perform commercial transactions on the Web. Functionality is a key word in this context, because it is now widely accepted that a web site is not merely a matter of aesthetics: correct functionality is required, linking the problem of developing web applications to the good practice required in Software Engineering. In consequence, a precise software production process is needed.

Nowadays, there are a number of initiatives which are intended to provide a solution for the creation of web applications within a well-defined software production process. Furthermore, these solutions must provide support for e-commerce due to the growth of commercial activities on the network. Several approaches to hypermedia design such as HDM [5], EORM [8], OOHDM [17], OOH-Method [6], W3I3 Tool Suite [2] and ADM [9] have been presented. Normally, hypermedia features and functional properties are treated separately, making it difficult to deal with the problem of developing a web application within a unified framework.

Address:
*Department of Information Systems and Computation, Valencia University of Technology,
Camino de Vera s/n, +34 96 387 7000, fax: +34 96 387 7359,46071 Valencia, Spain.
E-mail: {opastor, sabrahao, jjfons}@dsic.upv.es

In practice, they provide only a partial solution because they focus either on hypermedia characteristics (focusing on how the navigation will be defined) or on more conventional characteristics (focusing on classes and operations to specify the system functionality). Beyond these partial solutions, it is beginning to be widely accepted that web sites are evolving from merely hypermedia information repositories to hypermedia distributed applications, currently referred to as web applications [1].

Our proposal provides a concrete contribution in this context: starting from the premise that conceptual modeling is needed in order to correctly develop web applications, we propose a different approach that focuses on the integration of navigational design and conceptual modeling. These can be used together as input for code generation environments. We have designed and implemented a software production process to put these ideas into practice. This can be called an e-modeling software production environment, meaning that we define a process for applying conceptual modeling techniques to the development of web applications.

In this paper, we specifically address the concepts related to e-commerce applications development. To properly deal with this problem, we integrate two activities which are traditionally performed separately: that of modeling the operations and that of modeling the hypermedia. In our approach, the conceptual modeling step is considered as a unique generic phase. Using what we could call a conventional OO conceptual modeling approach, the needed expressiveness is introduced in the model in order to properly specify navigation and presentation features. All this information is used to provide a precise methodological guidance for going from the conceptual space to the solution space (represented by the final software product).

The method taken as the basis for this process is the OO-Method [11] [12]. The OO-Method collects the system properties which are considered relevant for building a formal, textual OO specification in an automated way. This formal OO specification constitutes a high-level system repository. Furthermore, the definition of a concise execution model and the mapping between the specification language and the execution model notions make it possible to build an operational implementation of a software production environment allowing for real automated prototyping. This is done by generating a complete system prototype (including static and dynamic characteristics) in the target software development environment.

In the context of the OO-Method project, efforts have been oriented towards the development of a new model to enrich the Object-Oriented Software Production Method with the required expressiveness to specify navigation features. This model has been called Navigational Model and provides navigational support for Object-Oriented Web-Solutions Modeling (OOWS) [16]. In order to model the desired system including the navigational aspects, the software production process is structured in two steps: Specifying the System and Developing the Solution. Following the OO-Method approach, a full specification of the user requirements is built in the Specifying the System step. Thus, a strategy oriented towards generating the software components that constitute the solution (the final software product) is defined in the Developing the Solution step. In this step, an e-commerce application, which is functionally equivalent to the specification, can be obtained in an automated way.

In accordance with these ideas, this paper is organized in five sections. Section 2 describes how an e-commerce application can be specified using conceptual modeling techniques. Special emphasis is placed on how navigational semantics are represented in the Navigational Model. Section 3 describes the “Developing the Solution step”, introducing the mechanisms used to generate the corresponding final software product. Section 4 presents a comparison with related works. Finally, section 5 provides concluding remarks and further work.

2. SPECIFYING THE SYSTEM

The problem peculiarities and the behaviour that the system must offer to satisfy the user requirements are identified in the Specifying the System step. This step includes the requirements collection based on a Use Case [7] approach and system conceptual modeling activities. When dealing with the conceptual modeling phase, the abstractions derived from the problem are specified in terms of their classes, structure, behaviour and functionality.

Historically, the OO-Method [11] uses well-known UML-compliant diagrams to represent the required conceptual information in three models: the Object Model, the Dynamic Model and the Functional Model. In this paper, a fourth model is introduced: the so-called Navigational Model. All together, these models describe the object society from four complementary points of view within a well-defined OO framework.

2.1 Conceptual Modeling

Conceptual modeling in OO-Method collects the information system relevant properties using three complementary models:

The **Object Model** that is represented by means of a Class Configuration Diagram, a graphic model where system classes are declared including their attributes and services. Aggregation and inheritance hierarchies are also depicted representing class relationships. Additionally, agent relationships are introduced to specify who can view the attributes and activate the services. The **Dynamic Model** that is used to specify valid object lives and interobjectual interaction. To describe valid object lives, we describe one State Transition Diagram (STD) for each class. To deal with object interaction, we use an Object Interaction Diagram (OID), for the entire System.

The **Functional Model** that captures the semantics associated to the changes of state of the objects motivated by the service occurrences. This model specifies the effect of an event on its relevant attributes through an interactive dialogue. The value of every attribute is modified depending on the action activated, the event arguments involved, and the current object state.

2.2 Navigation Modeling

The navigation semantics associated to the system users is specified, starting from the classes of the Object Model. This added expressiveness is the core of what we call the OOWS approach [13]. In OOWS, the web-solution application is obtained by adding a navigational view over the OO-Method Object Model. The navigational semantics of an e-commerce application is captured based on the point of view of each agent identified in the object model. This semantics is described in a **Navigational Model**, using a UML-like [3] notation.

A navigational model is essentially composed of a navigational map (see Figure 1) that represents the global view of the system for an agent in OO-Method. It is represented by a directed graph where nodes are navigational contexts and arcs are navigational links. A **navigational context** represents the point of view that a user has on a subset of the object model. A **navigational link** allows navigating from one context to another.

During the definition of a navigation map for an Agent, it is possible to add some expressiveness to control the *session* for the agent. The services that will be executed at the beginning of the session (delimited by '#') and at the end of the session (delimited by '##') can be defined. This information is usually specified as a textual description for a navigational map (see Figure 1). For instance, in a store e-commerce application, when a request from a new anonymous user (Agent) is received, the system could execute a service to create an instance of this agent class and a service to assign him a shopping cart. When the system detects that this agent has finished its session, a service to eliminate the shopping cart for unconfirmed purchases and a service to eliminate this agent instance will be executed.

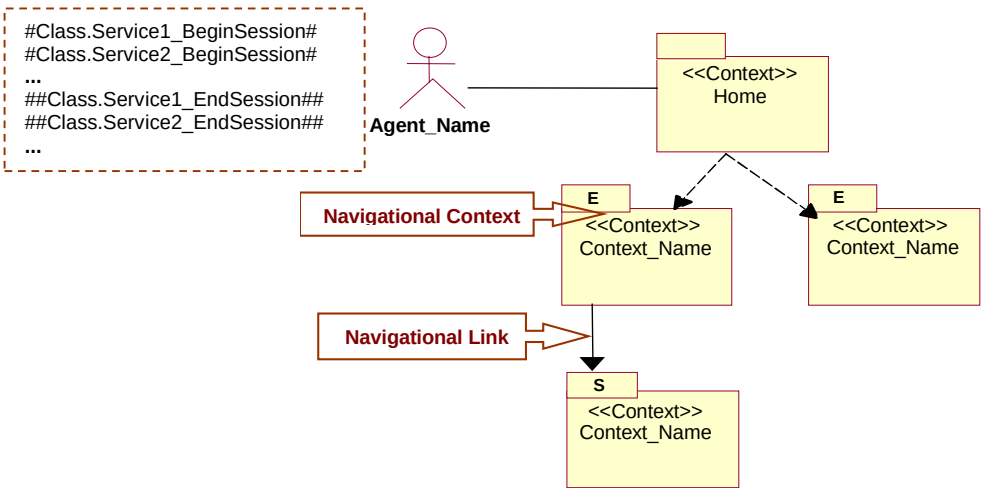

Fig 1. A Navigational Map

Basically, a navigational context is composed of three areas: *Context Definition Area*, *Navigational Area* and *Advanced Features Area*. Each area is described using basic primitives. The adopted notation for these primitives is shown in Figure 2. Next, we are going to explain these areas in detail.

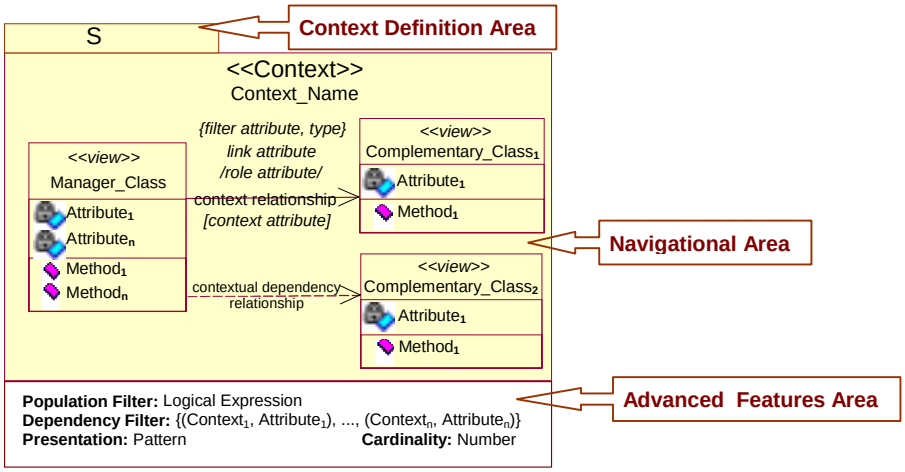

Fig 2. A Navigational Context

Context Definition Area

The context definition area shows the type of context. Navigational contexts can be classified into two types: **exploration** contexts (E) and **sequence** contexts (S). The exploration contexts can be reached at any moment independently of the current context. The sequence contexts can only be reached following a predefined sequence of navigation links.

Navigational Area

The navigational area is composed by a set of classes stereotyped with the reserved word «view». These classes are called navigational classes and can be connected by two types of relationships. In fact, a navigational class can be defined as a projection (view) specified on a class of the object model which includes a given subset of its attributes and operations.

For each navigational context there is a main class that is called **manager class** from where navigation starts. The others classes are called **complementary classes** and they contribute to giving additional information to instances of the manager class. The types of relationships that can be defined between navigational classes are context relationships and contextual dependency relationships.

A **context relationship** is a unidirectional binary relationship that can be defined on existing aggregation or inheritance class relationships. Once again, these aggregation or inheritance relationships come from the object model, assuring full compatibility between the navigation model and the other models used in the process of conceptual modeling. A context relationship indicates the navigation direction between the corresponding classes. Graphically they are represented using solid arrows.

In a **contextual dependency relationship**, the navigational semantics towards the target class is not defined: this kind of relationship is used to provide the required additional information in the current node without denoting any further navigation. Thus, the existence of an associated navigational context is not necessary. Graphically it is represented using dashed arrows.

In a context relationship, context attributes, link attributes, filter attributes and role attributes can be specified. We now introduce these four primitives in detail. A **context attribute** specifies the target navigational context of a navigation link. A **link attribute** specifies which attribute of the target class is involved in the connection defined by the corresponding navigational link. A **filter attribute** introduces a selection criteria on the population of the target class, based on the value of the involved attribute. In this way, the search for specific objects is made easier. It is possible to specify the following basic behaviour for a filter: exact (E), approximated (A) or range of values (R).

- **exact:** the instances of the manager class in the target context will have the exact value introduced by the user.
- **approximated:** the instances of the manager class in the target context will have the approximated (like) value introduced by the user.
- **range:** the instances of the manager class in the target context will be between the limits of the value introduced by the user.

A **role attribute** indicates the role that makes reference to the relation when two classes have more than one relationship. In these cases, the name of the target class cannot be used to identify the relationship and to navigate unambiguously, and the role attribute provides the solution.

Finally, **service links** can be defined for the services of the classes. A service link is associated to a target navigation context and it means that the execution of that service will produce a jump to the associated navigational context.

Advanced Features Area

In a navigational context, a **population filter** for the manager class can be established during the design. It is represented by a logical expression on the corresponding class attributes. A **dependency filter** shows a tuple (context, attribute) for each context ending in the current context. Furthermore, it is possible to specify the **presentation** style of the information using the following patterns: *register mode*, *tabulate mode* and *tabulate master-detail* mode. Finally, the exploration **cardinality** fixes the number of instances of the manager class (with its corresponding dependencies) that will be shown in the context. These features are defined in the lower part of a navigational context (see Figure 1).

It is possible to completely specify the semantics attached to the navigation requirements for web-oriented applications using the primitives presented above. An (necessarily brief) example that includes the above comments is shown in Figure 3. This example shows how a generic Shopping Cart of an e-commerce application can be specified using the OOWS approach. A section of Object Model related to the Shopping Cart class and its relationships is shown on the left side of Figure 3. A navigational context (ShoppingCart_Details) based on this object model is described on the right side.

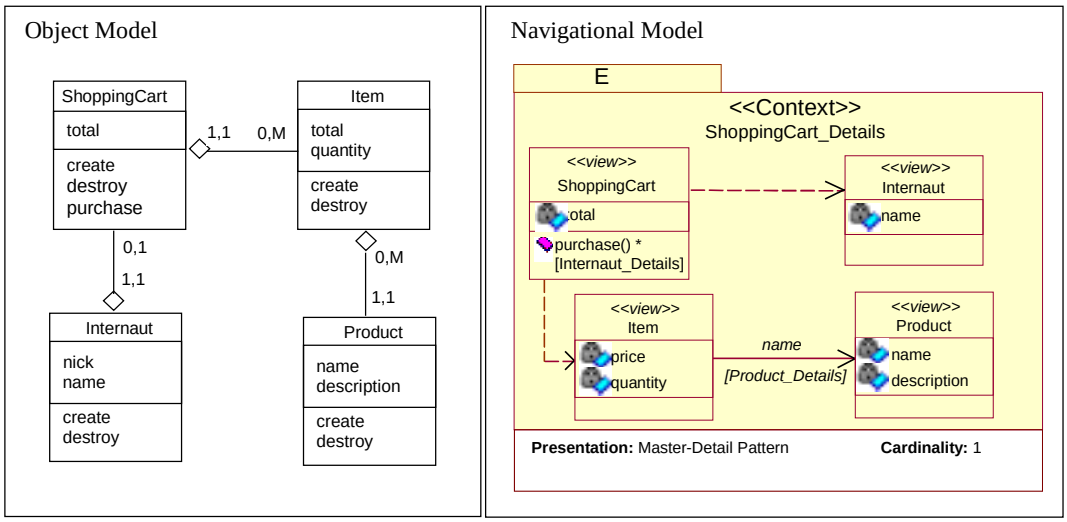

Figure 3. An example of a generic Shopping Cart

The ShoppingCart_Details context describes all the products selected by an Internaut. This context is defined as an exploration context. The navigation starts in the ShoppingCart class that acts as the manager class. The *total* attribute from the Shopping Cart class will be shown. The *purchase* service allows users to perform a sale. This is a service link that is associated to the *Internaut_Details* context. When the service is executed, a navigation to this target context is produced.

The other classes are complementary classes and they provide additional information related to the ShoppingCart class. In order to provide this information, the ShoppingCart class has a contextual dependency relationship with them. However, no navigation is defined by any

relationships of this kind. For instance, the relationship from ShoppingCart class to Item class only presents the *price* and *quantity* attributes of a ShoppingCart. Additionally, a contextual relationship appears. From an Item, internauts can obtain more details of a Product (Product_Details context) by selecting its *name*.

Once the conceptual model has been completed (including the navigational information), a full web-based application can be automatically generated in the next step ("Developing the Solution" step), according to the OO-Method approach that reifies the conceptual model in the target development environment.

3. DEVELOPING THE SOLUTION

The generating of the software components corresponding to the conceptual constructs used in the conceptual modeling step is accomplished in the Developing the Solution step. A precise definition of these mappings between conceptual constructs and software representations make it possible to generate the final software product in an automated way. This is especially important at a time when the need for providing correct e-commerce applications is continuously growing. We argue that to assure correctness, the final software product must correspond in a precise way to the conceptual schema collected in the specification phase. To reach this objective, the approach presented here uses a structure which is composed of three layers: Data, Business Logic and Presentation.

The **Data** layer is represented by the relational database that stores the population of each class that has been identified during the specifying step. A relational schema of a database can be obtained using the information described in the Object Model. Then, the definition of this schema can be automatically obtained using a specific DBMS. A persistence class to retrieve/update the instances of the classes from/to the database that will be used is also defined in this layer. This class offers an interface that isolates the components of the application from the physical support. Moreover, this class also provides other services such as: creating and deleting instances, transaction management and other auxiliary ones.

The **Business Logic** layer contains the logic of the business for each class specified in the conceptual model. This layer acts as the system middleware to support the integration of the functionality and the data management. An interface that implements the OO-Method classes with the definition of the common services is defined. A service manager is used to begin the execution of these services, including transaction management, integrity constraints and trigger verification.

The **Presentation** layer contains all the presentation logic of the application. According to the proposed strategy, (HTML, DHTML, XML, WML, etc.) server pages with server code embedded (such as ASP, JSP, ColdFusion, Php, etc.) could be generated from the navigational model. These pages use the Data layer to retrieve information for generating the client graphical user interface, and they use the middleware provided by the Business Logic layer for executing services. In order to generate the Presentation layer, we proposed a mapping from OOWS primitives to web application software components. This mapping is structured according to primitives introduces in the previous section: navigational context, navigational classes, filters, context relationship, contextual dependency relationship, services, service links, read/write attributes and presentation pattern.

A server page is created for each *navigational context*. All the specified information (navigational classes, filters, navigational links, presentation, etc.) will be managed inside this page. A link for each exploration context will appear. The *navigational classes* are instances of the classes implemented in the business logic layer. It is necessary to declare a manager class with all the instances that fulfill the defined filters. The complementary classes are obtained from the navigation functions by using the corresponding roles defined in the business logic layer. The attribute values of the classes are obtained from the properties that have been defined in the business logic layer.

The *dependency* and *context filters* are used to retrieve the population of the manager class. Their use provides the view of the objects which are actually alive in the system. The filters of the complementary classes are used to retrieve the instances of the class using the corresponding navigation functions. If a *context relationship* contains a link attribute, the server page must create a link for each value of this attribute. This link enables the navigation to the target context, filtering it with the selected object. If the context relationship does not have a link attribute, a link will have to appear in the server page showing the alias of the target context. This alias represents

the selection of all objects in the target context through the structural relationship (defined in the Object Model).

When a *filter attribute* appears in a context relationship, the required information in the source context depends on the filter type (exact, approximated and range). This filter will have to be executed in the target context when the population of the manager class is retrieved. A *contextual dependency relationship* is used to join two navigational classes in order to refine a structural relationship. It does not have any associated functionality, so no mapping pattern is required. A *service page* is created for each service that appears in a service link. To execute the service, a service page uses the middleware objects of the business logic layer. These pages ask the user for the required parameters of the service. When a *service link* appears, a link to its service page is created. Finally, the *presentation pattern* allows for different presentations of the information associated with each navigational context. A "template" is specified for each kind of presentation (register mode, tabulate mode and master-detail mode).

Due to current space limitations, detailed information about these patterns can be found in [15]. The results of these mapping patterns provide a web-based application. We are conducting experiments in order to apply this strategy to validate the OOWS approach for developing e-commerce applications. These experiments include an application for ticket sales of a theatre company [16] and a music website [14]. The architecture chosen for these case studies was Visual Basic v6.0 as implementation language together with the Internet Information Server with technology ASP (IIS/ASP) as web server. The results have been very impressive, due to the fact that the system must be specified before facing any implementation features. In consequence, the benefits are twofold: on the one hand, a sound system specification is provided: on the other hand, there is a precise methodological guidance to assure that the final software product corresponds to the system specification, according to the previous strategy.

4. COMPARISON WITH RELATED WORKS

The comparison of OOWS with other approaches (HDM [5], ADM [9], OOH-Method [6] and OOHDM [17]) is structured according to the following points of view: conceptual design, navigational design, interface design and generation of the application.

In the *conceptual design* step, the HDM approach is based on E-R diagrams and, therefore, does not use the OO paradigm; Araneus (ADM) is not based on a conceptual model, since its methodology depends directly on the navigational model; OOHDM uses an OO conceptual model based on UML with some modifications, but it does not permit objects to offer services to the users of the application. Finally, both OOWS and OOH-Method use OO-Method as conceptual model.

In the *navigational design* step, all proposals share the fact that they introduce node and link concepts as the basic structures for the description of hypermedial systems. However, they are considered in a different way. These approaches can be categorized in two groups: the first group is represented by HDM and Araneus and the second one by OOWS, OOHDM and OOH-Method. The former do not make a distinction between the conceptual and navigational models, but represent navigational characteristics in a conceptual model based on E-R design. The latter distinguish between an OO conceptual model and a navigational model, which specifies the view of each user of the system.

Within this group, there is a new difference: a navigational unit in OOWS and OOHDM (navigational context and node, respectively) is an element composed by several classes from the conceptual model. Whereas, in OOH-Method, the navigational unit is a class from the conceptual model (navigational class). Additionally, an element (navigational target) is used to group classes. It does not imply any navigation; it is merely used for organizational purposes. This concept is also defined in OOHDM approach as navigational context. The main difference between OOWS and OOH-Method is the concept of node. Whereas nodes (navigation classes) in OOH-Method are limited to presenting information of only one class, nodes in OOWS (navigational contexts) can work with views of several classes, showing more appropriate information for the user at every moment. This assumes that OOWS nodes reduces the number of jumps and allows the user to have a global view, instead of providing partial solutions.

Finally, in this section we comment how these approaches deal with the inclusion of the dynamic behavior in their models. Neither HDM nor OOHDM make any mention about the possibility of including services in their systems. The only dynamic behavior that they offer is queries to databases and generation of dynamic pages. When multivaluated attributes are defined,

Araneus allows the inclusion of the service that must be executed. Both OOWS and OOH-Method allow the inclusion of the services of each class that will be used in their navigational diagram.

In OOWS, the valid services that can be included in the navigation classes are only those that the user agent is allowed to activate, according to the conceptual model specification, whereas in OOH-Method it is necessary to indicate which services can be executed using service links. The approaches above mentioned deal with *interface design* in different ways. Currently, we are using OOWS together with an interface model for OO-Method [10]. OOHDM uses an interface model that designs Abstract Data Views (ADV) that is used to describe the objects perceived by the user. In Araneus, the interface is not designed, but it is possible to use style sheets to generate HTML HDM pages.

One of the strongest motivations that have promoted the development of these approaches is the *generation of the application.* A great part of the work invested in the different design steps of the system would be useless if automatic methods to generate whole or part of the system are not provided. The OOWS approach as well as OOH-Method proposes an automatic generation of the web application from OO-Method conceptual models. Nevertheless, the generation that the other methodologies propose has some peculiarities: HDM and OOHDM generate applications that do not have any dynamic part. These systems presents information based on Database queries, but do not offer services to the agents of the system. Finally, the applications generated by Araneus offer services through HTML pages using CGIs that must be separately written, although current work is being done to automatically generating some partial Java code. The generated applications depend on the Araneus system to be executed, because some parts are included in the final software architecture.

In summary, the main contributions of the OOWS proposal are two: first, the process of conceptual modeling is completely embedded in the system conceptual modeling step, because the new navigational model extends the preexistent expressiveness in a natural way. Second, the way to the e-commerce final software product (from the problem space to the solution space) is methodologically guided by an execution strategy that establishes how to properly reificate conceptual constructs into their corresponding software representations, opening the way to an automated implementation.

5. CONCLUDING REMARKS AND FUTURE DEVELOPMENT

In this paper, we have presented a software production process for e-commerce applications development. This is based on object-oriented conceptual modeling techniques applied to the development of web applications. Primitives to capture navigational requirements of the e-commerce applications have been presented. Then, we explain how these primitives can be mapped in a solution for automating the software development process. Currently, we are applying this approach to several real-world e-commerce applications including the Amazon.com electronic shop and a Forum application. The experience gained has allowed us to put into practice the OOWS approach and to refine the proposed primitives. Also, a strategy for the component generation oriented to an automatic implementation of the system was defined.

The process introduced in this proposal structures in a precise way the activities that must be undertaken in order to have an e-commerce application from an OO Conceptual Schema including navigational capabilities. This is an important advantage because developers know what to do and in which order, within a well-defined software production process specially designed to deal with e-commerce applications, and assuring that the final implementation represents adequately the system properties captured in the source Conceptual Schema. As the OO-Method provides model-based code generation capabilities, the final software product can be obtained in an automated way, improving the efficiency and the effectiveness of the software production process as a whole.

Current research work is being developed in this context to determine how to add a capability to measure in detail such improvements when compared with other process proposals of the area, based on the experience that we are obtaining in the development of complex web applications. In addition, we are building a translator based on XML [4] in order to support multi-device outputs. The future development involves the extension of the modeling language to specify security features and integrity validation of the navigational model with respect to the other elements of the conceptual model.

ACKNOWLEDGEMENT

We would like to thank Sergio Ramón for his help and support in implementing case studies to apply the proposal software production process.

REFERENCES

1. Baresi L., Garzotto F., Paolini P. From Web Sites to Web Applications: New Issues for Conceptual Modeling. *ER'2000 Workshop on Conceptual Modeling and the Web*, LNCS 1921. Springer-Verlag, 2000, pp. 89-100.
2. Bonifati A., Ceri S., Fraternali P., *et al.* Building Multi-device, Content-Centric Applications Using WebML and the W3I3 Tool Suite, *In Proc. 19th International Conference on Conceptual Modeling*, Salt Lake City, USA, 2000, pp. 64-75.
3. Booch G., Jacobson I., Rumbaugh J. *The UML Language Users Guide*. Addison-Wesley, ACM Press, 1999.
4. Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/1998/REC-xml-19980210, 1998.
5. Garzotto F., Paolini P., Schwabe D. HDM - A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, 11(1), 1-26, 1993.
6. Gómez J., Cachero C., Pastor O. Extending a Conceptual Modeling Approach to Web Application Design. *In Proc. Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS 1789, Springer- Verlag, 2000, pp. 79-93.
7. Jacobson I., Christerson M., Jonsson P., Overgaard G. *OO Software Engineering , a Use Case Driven Approach*. Reading, Massachusetts. Addison-Wesley, 1992.
8. Lange D. An Object-Oriented Design Method for Hypermedia Information Systems. *In Proc. Hawaii International Conference on System Science*, number 27, January 1994.
9. Mecca G., Merialdo P., Atzeni P., Crescenzi V. The Araneus Guide to Web-Site Development. *Technical Report, University of Roma*. 1999.
10. Molina P., Pastor O., Martí S., Fons J., Insfrán E. Specifying Conceptual Interface Patterns in a Object-Oriented Method with Automatic Code Generation. *2nd International Workshop on User Interfaces to Data Intensive Systems (UIDIS2001)*, Zurich, Switzerland, 2001. (to appear)
11. Pastor O., Insfrán E., *et. al.* OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods. *In Proc. Conference on Advanced Information Systems Engineering (CAiSE'97)*, LNCS 1250, Springer- Verlag, 1997, pp. 145-159.
12. Pastor O., Pelechano V. *et al.* From Object Oriented Conceptual Modeling to Automated Programming in Java. *In Proc. International Conference on the Entity Relationship Approach (ER'98)*, LNCS 1507, Springer- Verlag 1998, pp. 183-196.
13. Pastor O., Abrahão S. M., Fons J. J. OOWS: An Object-Oriented Approach for Web-Solutions Modeling. *In Proc. Media in Information Society (MEIS'00)*, Ljubljana Slovenia, 2000, pp.127-128.
14. Pastor O., Fons, J. J., Abrahão S. M., Ramón S. Object-Oriented Conceptual Models for Web Applications. *In Proc. 4th Iberoamerican Workshop on Requirements Engineering and Software Environments (IDEAS'01)*, San Juan, Costa Rica, 2001, pp. 239-251. (in Spanish).
15. Pastor O., Abrahão S. M., Fons J. J, Ramón S. Transforming conceptual patterns into software components for e-commerce applications. *Technical Report, OOM Group 01-2001*, DSIC-UPV, Valencia, Spain. (in Spanish)
16. Pastor O., Abrahão S. M., Fons J. J., Ramón S. An Object-Oriented Approach to Automate Web Applications Development. *In Proc. 2nd International Conference on Electronic Commerce and Web Technologies (EC-WEB'01)*, LNCS, Spring-Verlag, Munich, Germany, 2001. (to appear)
17. Schwabe D., Rossi G. The Object-Oriented Hypermedia Design Model. *Communications of the ACM* 38(8), 45-46, 1995.