The Dynamic Business Object Pattern

Russ Rubis
Dept. of Computer Science and Eng.
Florida Atlantic University
Boca Raton, FL 33431
rrubis@fau.edu

Dr. Ionut Cardei
Dept. of Computer Science and Eng.
Florida Atlantic Universit
Boca Raton, FL 33431
icardei@fau.edu

ABSTRACT

A business object is an object which is used often by business applications and services and is a widely acceptable entity in the running of the business. For example, a purchase order, an invoice, and a customer profile could be considered business objects, as these are parts of day-to-day business activity, but more importantly these have well-established and commonly accepted attributes and behavior. Developing business objects from scratch each and every time one is needed is a resource-intensive, highly repetitive, and unnecessary undertaking. One of the challenges faced by today's enterprise application developers is the lack of a generic approach for specifying the design, delivery, and processing of business objects. In general, a business object may contain one or more of the following: dynamic data object, static document, and workflow. Each time a custom application requires the use of a business object, the developers either have to design it themselves, or use some language and/or platform and/or software dependent solution/library which might or might not address the needs of their application. This paper proposes a new pattern for specifying Dynamic Business Objects for business applications. The objective of this pattern is to provide a generic approach to design extensible Business Objects and their frameworks for business applications.

Keywords

Business data object, business document, workflow, common business object, dynamic object

1. INTRODUCTION AND OVERVIEW

Just about every business is comprised of multiple organizations and or departments, each specializing in their area of function. An accounting department deals with various customers, distributors, and partner accounts, the procurement department handles purchase orders, invoices to procure goods and services for the business, while the warehouse receives and distributes the goods. Each organization or department needs applications specific to their functional area. These applications are implemented on a web-based infrastructure using standard web technologies and rely on "business objects" such as purchase orders, invoices, and accounts. These business objects have a common set of properties at the design level and different implementation solutions [.NET, J2EE, Spring, CORBA, etc.] as well as custom architectures [Ariba, SAP, SmartClient, etc.]. The business objects reveal similarity in the overall approach without a systematic effort to build a reusable specification and infrastructure.

This paper introduces the Dynamic Business Object Pattern , our attempt to address specification, design, and code reuse when building business objects for specific problem domains.

A Dynamic Business Object refers to an abstract generalization of these various application specific objects that provides a uniform and reusable specification. The Dynamic Business Object Pattern describes a generic and systematic approach to organize the business objects and to describe them. It structures business objects in static documents, workflows, and dynamic data. Business objects have the typical properties of object-oriented systems, such as composition, inheritance, polymorphism, plus they promote reuse at several levels.

For example, a generic Accounting Module aggregates business objects involved in supporting accounting applications. A typical Accounting Module could contain Dynamic Business Object such as General Ledger, Project Account, Expense Account and others. The module's business object specification includes the basic elements necessary to create, update and maintain the data, and functions needed to perform accounting tasks for a business. Business objects from this Accounting Module can be extended to accommodate specific attributes and behaviors customized for an individual company or industry. When a new module is extended from an existing one, it inherits all of the parent module's objects and attributes, including dynamic data objects, static documents, and workflows.

The Dynamic Business Object Pattern also describes an architecture to implement and support business objects. For example, a dynamic data object implies that a storage solution is required; it also implies that a custom user interface will be necessary to present the dynamic data object. A static document, on the other hand, is assumed to have a widely accepted form of viewing software (i.e. MS Word, PDF, Notepad, etc.). Finally, a workflow is a process which supports the life cycle of a dynamic business object.

Another benefit of a modularized approach is that each existing module is not static, and can be enhanced over time. For example, the Common Module (see below) has three main parts: Dynamic Data Object, Static Document, and Workflow. If another entity needs to be added to the Common Module in the future, it can be done so safely, without affecting the existing functionality.

There are a number of patterns that deal with web applications and web object specifications and architectures. Among them are Document-View-Presentation [8], Model/View/Controller [9], Presentation/ Abstraction / Control [10], and others. In addition to these, there are also analysis patterns which address specific business domains [4]. The Dynamic Business Object Pattern is a new approach to abstracting business objects using the three basic elements: dynamic data objects, static documents, and workflows.

The intended audience for this article includes business application architects and developers at the early stage of Dynamic Business Object design.

2. THE DYNAMIC BUSINESS OBJECT PATTERN

2.1 Intent

The Dynamic Business Object Pattern describes an extensible design for business objects used for business applications and a structure for their platform-neutral specification.

2.2 Example

Consider a simple customer request process, which consists of a customer initiating a request, and the request being received and processed by a a business. Lastly, notification of completion is sent to the customer and is the final step in the customer request process.

The abstract customer request process can now be extended to formulate a more complex example reflecting a specific business process. Consider a shopping cart. At its base, it is a customer request for goods or services performed using a web based application. After selecting items and adding them to a shopping cart, the customer provides the billing and shipping information, then submits the request for processing. The vendor then receives the shopping cart request, verifies the billing information, fills out the order and ships it to the customer. Finally, the vendor notifies the customer via email that the shopping cart request has been completed.

2.3 Context

Almost all businesses have a web presence for marketing, education, and sales. Many businesses run (directly or via service providers) web-based applications that encompass most business processes, such as customer relations, product support, sales, etc. Businesses require flexible web solutions and reliable frameworks to support their day-to-day operations. The object-oriented design methodology is widely used in the software industry to model data and logic for business applications. Business objects should have the flexibility and simplicity to solve diverse business problems.

Businesses also require a unified approach to designing and building business objects. Developers of commercial application frameworks and custom web applications benefit from a portable and reusable business object specification.

The Dynamic Business Object Pattern could also be applied to environments that do not involve distributed computing. However, we believe that it is best suited for web-based, distributed and heterogeneous computing architectures with a variety of client devices (PCs, smartphones) and service platforms.

2.4 Problem

Business applications involve complex distributed operations orchestrated on client and server side. Applications rely on a variety of documents with static and dynamic content for presentation and for data storage. Document structure and application behavior require a platform-neutral and reusable specification that can be specialized further for particular business cases. The specification must exhibit object-oriented properties, such as modularization, inheritance, and polymorphism. The object specification must include data, format (presentation), behavior, and various constraints. Business objects must be based on proven business practices. At the same time, business objects should have the flexibility to alter their behavior based on the changing needs of a business.

2.5 Solution

The Dynamic Business Object Pattern presents an approach for specification and design of business objects. The pattern's top-level specification element is the Common Module (see Figure 1), comprised of three main elements that are the minimum needed to support the description and behavior of a dynamic business object:

1. Dynamic Data Object
2. Static Document
3. Workflow

A dynamic business object specification may include any combination of the above elements. For example, a shopping cart contains items that the user wishes to order (incidentally, these items are also dynamic data objects). The shopping cart item might have an image of the product (an image being a static document). After the shopping cart is submitted by the user, it is processed by some predefined workflow, which ensures that the shopping cart is filled and the ordered items are shipped to the user. Although the above example is simplified for the purpose of clarity, arguably just about every business object can be presented in the pattern using the three elements introduced in the Common Module: dynamic data object, static document, and workflow.

Through extension, developers and framework designers can build a library of modules derived from the Common Module. These modules are generic at the top of the hierarchy and then get more specific to industries and businesses lower in the hierarchy. Modules can be combined to design sophisticated business applications.

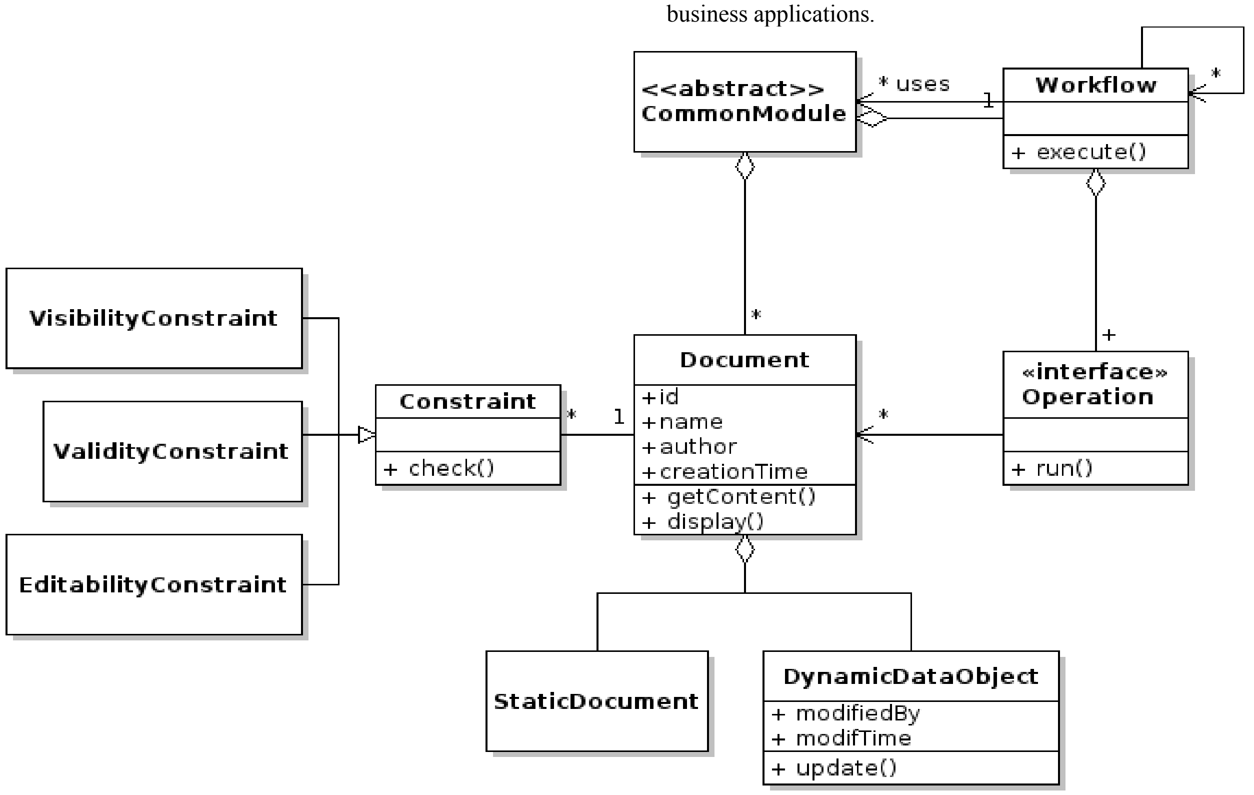

Figure 1. The block diagram structure of a dynamic business object.

2.5.1 Dynamic Data Object (DDO)

This object represents the data associated with a Dynamic Business Object that could change over time as a result of some business process or user input. A good example would be a customer data. New customers are often created, and existing customers are frequently updated (i.e. address, contact, payment terms, etc.). A dynamic data object can be maintained via a user interface, or a business process (i.e. a customer data feed is received nightly). The important distinction of a dynamic data object is that, unlike a static document, its contents can be (and often are) changed, and unlike a workflow, it is not a process, but rather an object (or an entity) representing a specific set of data. For example, a purchase order object contains a set of elements representing the items being purchased, from whom, by whom, and under what terms. A purchase order workflow, on the other hand, is a process which takes place from the time a purchase request is submitted, to the time the purchased items are delivered to the buyer (more on the workflow later). Another distinction of a dynamic data object is that, unlike static document, it requires a custom storage solution (i.e. database), as well as a custom user interface. For example, a purchase order entry screen, with its validations and constraints, is a custom interface linking the purchase order data between the user creating it and the data storage where it will be persisted. Even if the data is not persisted in the database, but in another storage (i.e. file, cloud, etc.), the format of the data record(s) must still be predetermined and agreed upon, thereby making it a custom storage solution.

The DDO in the Common Module is an abstract object type and is assumed to contain a limited set of attributes within the Common Module. These fields are the object's internal ID, the timestamp of creation and update, as well as the fields identifying who created the object and the last user id which updated the object, if applicable. Within the Dynamic Business Object Pattern, these fields are automatically inherited by all dynamic data objects which extend from the Common Module's Dynamic Data Object.

A DDO can contain other dynamic data objects, as well as static documents. For example, a shopping cart (a DDO object) contains one or more shopping cart items (a shopping cart item is also a DDO object). A shopping cart item may include an image of the item, where image is a static document.

The DDO employs a set of constraints to control what access rules and specific behaviors are permitted on the object and how they should be accomplished. These constraints must be specified using UML's Object Constraint Language where possible or explicitly, with logic in sequence or other behavioral UML diagrams.

2.5.2 Static Document

A Static Document within the Dynamic Business Object Pattern is any document whose data cannot be changed by end users (i.e. clients) of the application. An example of a static document could be a report printout or a product image file, which users (clients) can view, but the contents of which cannot be modified by clients. Within the Common Module, static documents can be generated from the dynamic data objects or can be imported from another system or environment. Because a static document cannot be updated, only the visibility constraint (Section 2.5.4.2) can be applied to it.

Unlike dynamic data objects, static documents are generally associated with a widely accepted client viewers, and do not require custom solutions for client presentation. For example, MS Word and Excel clients are used to view Word and Excel documents respectively, while Adobe Acrobat is used to view PDF documents.

Within the Common Module the static document can be divided into 2 groups: print media and electronic media. As the physical print is still widely used by businesses, it should not be ignored.

2.5.3 The Document Superclass

The Document class factors out common attributes and behaviors common to dynamic and static documents.

2.5.4 Constraints

The Constraints provide a way to limit the access to the dynamic data object, and to some extend to the static data object as well. The Dynamic Business Object Pattern includes three constraints which can be imposed on the object: visibility constraint, validity constraint, and editability constraint.

2.5.4.1 The Visibility Constraint

The Visibility Constraint (ViC) contains logic that dictates the visibility of a component or attribute of that object. The ViC is not required for every object or entity; it should only be used when needed by the application logic. For example, a *ship-to* component of a Shopping Cart business object should always be displayed on the shopping cart UI, as without ship-to, there is no way to learn where the shopping cart items will have to be delivered. Therefore, by default, there is no need for a ViC on a *ship-to* component. However, a *tax* on a shopping cart is not always required, and thus would be an ideal candidate for ViC. In this case, the visibility constraint might contain a logic which states that items shipped to certain states need to collect a tax, therefore the tax field must be displayed on those orders. Similarly, the visibility constraint logic would hide the tax fields on orders which are shipped to states which do not require sales tax collection.

2.5.4.2 The Validity Constraint

The Validity Constraint (VaC) determines the validity of an element (sub-object or attribute) of a DDO. A VaC is not required for every object or entity by default. For example, a shopping cart's *comment field*, being optional, is valid whether it is empty or contains a comment text. However, a *ship-to* is required before the shopping cart can be submitted for processing. Validity constraints should only be used when a more complex logic is required to determine if an object or its entity is valid or not.

The VaCs can also be employed by workflows to perform further validation. For example, a workflow could perform a validation on a field to check for malicious entrees (i.e. links to spam or inappropriate URL, cross scripting, SQL injections, etc).

VaCs should not be used in cases where the constraint can be expressed using established UML notation.

2.5.4.3 The Editability Constraint

The Editability Constraint (EC) contains logic that determines whether an element (component or attribute) of a DDO is editable, i.e. changeable by some user interface. The EC is not required for every object or entity; it should only be used when editability of an element is driven by business logic. For example, a Shopping Cart *comment field* should always be editable as it is intended to gather additional information from the user. However, a tax field or a shipping charge field should not be editable because these are derived based on item cost and shipping destination. The EC comes into play when the editability of an object or its entity depends on some additional logic which must be recorded as part of the analysis and design process. Element editability can be qualified (parameterized) with contextual information, such as *user role*. For example, a clerk at a warehouse could verify the quantity of an item which needs to be shipped, however only a warehouse manager can edit that quantity if needed.

The use of roles can also be extended to the Visibility and Validity constraints by applying Securing Analysis Patterns [2].

It is important to stress that constraint classes do not alter the structure of the object itself or of its entities. If, for example, a Tax element is hidden by ViC, its value remains unchanged and could not be (nor should be) altered by any of the constraints.

2.5.5 Workflow

The Workflow component describes the logic associated with the Dynamic Business Object, the sequence of operations that the Dynamic Business Object goes through during its lifetime. A workflow's operations are distributed on the client and on the server side and can be executed sequentially or concurrently. A Dynamic Business Object's workflow can include other sub-workflows to modularize complex behavior. The runtime for a workflow is platform-neutral, generic, and and is interpreted by the various actors involved in its execution.

A workflow involves the Dynamic Business Object's dynamic and static documents, and possibly other business objects, as well as services provided by the infrastructure.

For example, the workflow of a Shopping Cart Dynamic Business Object begins when customer submits the shopping cart with desired items to be purchased. The workflow then executes logic which verifies customer's payment information, then forwards the shopping cart request to the warehouse for packaging and shipping. Once items are shipped, the workflow will notify customer of completion.

Workflow is an essential part of the Dynamic Business Object Pattern as its main responsibility is to track the status of the business object through its life cycle, and take appropriate action in each stage. For example, when user submits a shopping cart to be processed, it is the shopping cart's workflow process that determines what needs to occur next. The workflow might first check the billing data of the shopper to ensure that it is valid. If could then check the warehouse inventory to ensure that the ordered items are available, and if they are not, the workflow would then send out an email to the shopper with further instructions. The workflow "moves" the business object from one stage to another, until the process is completed.

Because the Common Module is the base module and does not represent a specific business process, its workflow includes just two trivial no-op operations, Start and Stop, that can be specialized in sub-modules (Figure 2).

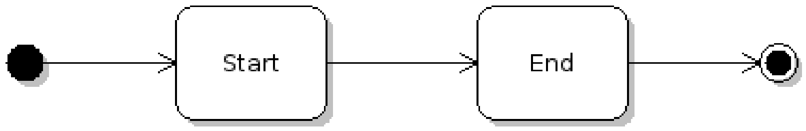

Figure 2. Default workflow for the Common Module.

3. KNOWN USES

Ariba Inc. uses similar approach in its architecture and software products [15]. Ariba uses the concept of an Approvable object, which is a representation of a business entity. The Approvable object is similar to the Dynamic Data Object presented in this paper. Within the Ariba architecture the Approvable object is coupled with the user interface and a storage medium, and its data is expected to be dynamic throughout the object's life cycle. The Ariba architecture also includes a workflow component which manages the Approvable object through the required business processes and events. Note: Ariba Inc. has recently been acquired by SAP AG [24].

4. IMPLEMENTATION: A SHOPPING CART

This example will show how a Dynamic Business Object Pattern employs an existing module to create (or extend) a new module to address a specific business need. In this example we'll create and apply pattern to a Shopping Cart Request Module. Our shopping cart module will be based on an existing module (Abstract Customer Request Module).

Every business has customers with whom it constantly interacts. One of the most generic interactions is to process customer requests. The requests can come in many different forms: an appointment, a question, a purchase of the business product or service, and many others. The most basic customer request can be broken down into the following steps:

1. Customer initiates a request
2. Request is received and processed by the business
3. Customer is informed of the request completion

Based on above, we can create a module called the Abstract Customer Request Module. Note that the Abstract Customer Request Module would have already been part of the Dynamic Business Object Pattern library, and is only detailed here for the purposes of illustrating how the Shopping Cart Module is eventually constructed.

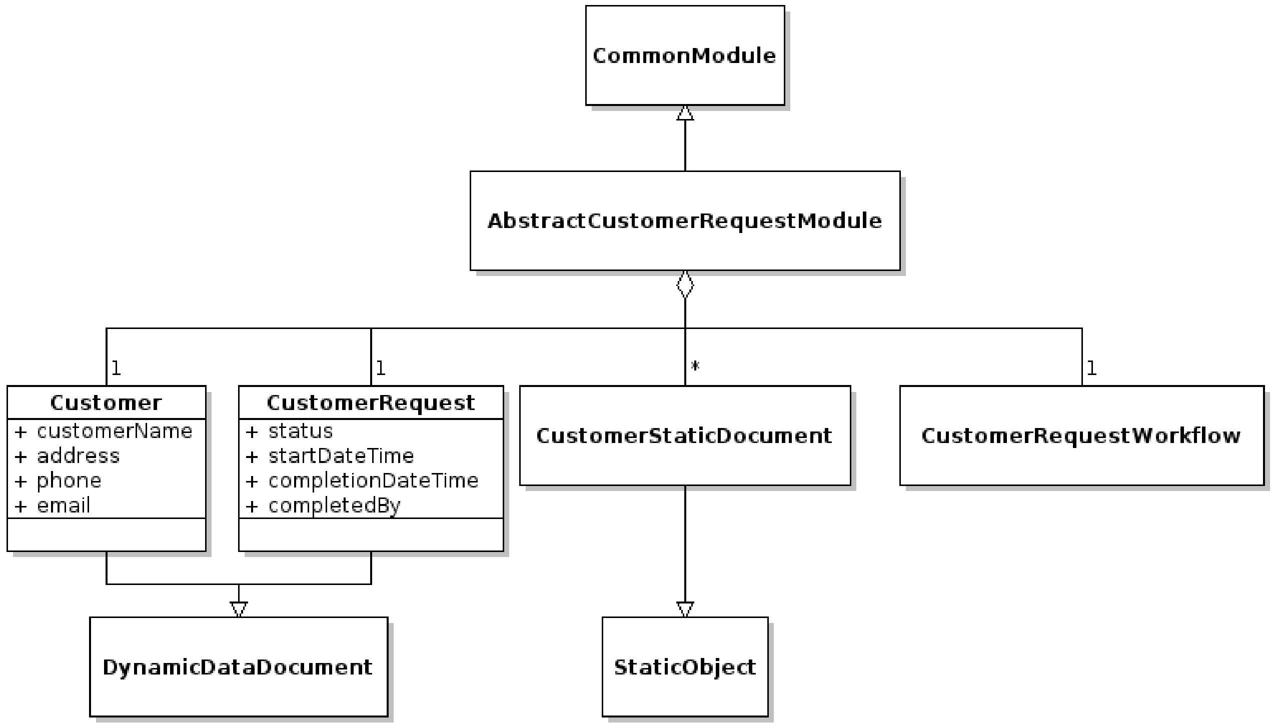

Figure 3. Diagram showing Common Module specialization for a Customer Request.

The Customer and Customer Request objects are an extension of the Dynamic Data Object found in the Common Module. Customer Static Document and Customer Request Workflow are an extension of Static Document and Workflow respectively, also found in the Common Module.

In addition to the fields inherited from the Dynamic Data Object, the Customer object also contains fields relevant to the customer's identity, while the Customer Request object gets additional fields relevant to the processing of the customer request.

There are no specific changes to the Customer Static Document object as it pertains to the Abstract Customer Request Module – the object is just inherited from the Common Module. The workflow in the ACR Module is modified to reflect the minimum needed to process a generic customer request, as seen in Figure 4.

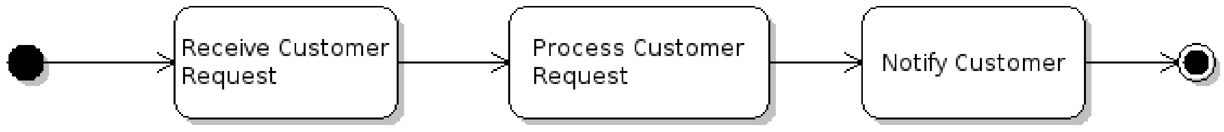

Figure 4. Abstract customer request workflow.

With the reusable Abstract Customer Request Module specification completed, the next step is to extend it according to the Dynamic Business Object Pattern to analyze and build the Shopping Cart Request Module.

In its most basis form, a Shopping Cart is a customer request for goods or services. A customer browses the catalog, selects items from it, and finally submits it for processing. For this reason, it only makes sense to rely on our ACR Module in order to build our Shopping Cart Request Module (SCR).

The ACR Module is extended for the new Shopping Cart Request Module, as shown in Figure 5. According to the Dynamic Business Object Pattern, the Shopping Cart Request Module inherits all objects found in the ACR Module. It is extended to reflect the nature of the shopping cart request: it is initiated by an online customer (hence the new login and credit card information) and new classes for the shopping cart and its items.

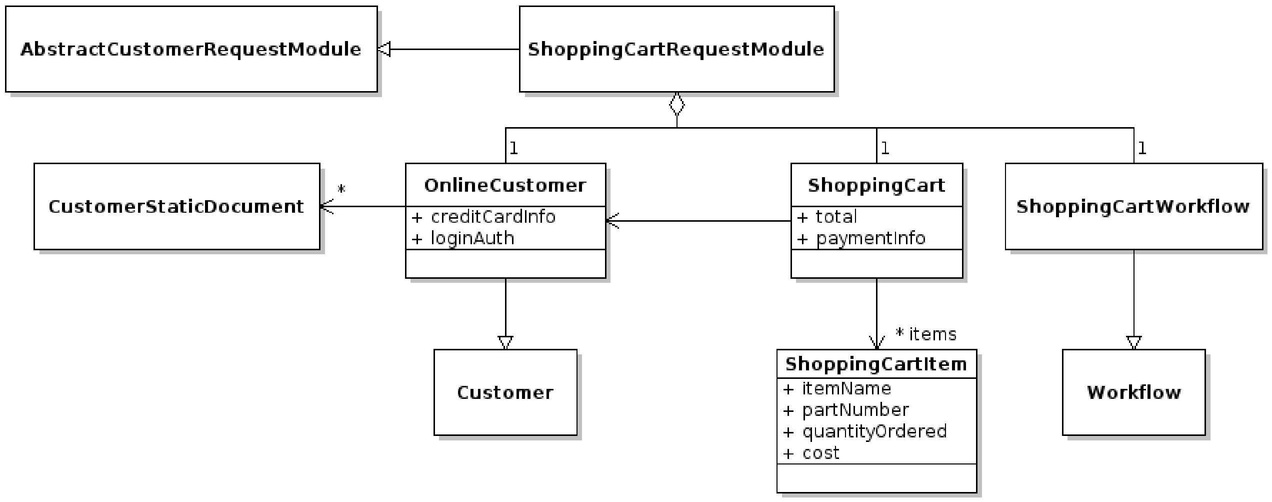

Figure 5. Shopping Cart Request Module specialization: support for online customers, shopping cart, and its items.

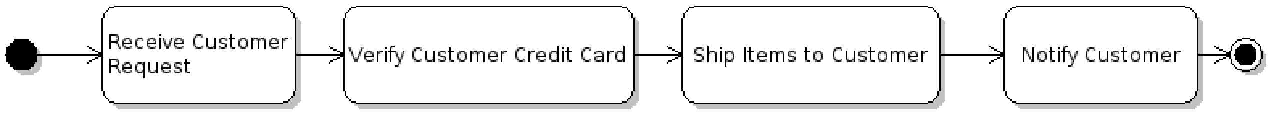

Figure 6. The workflow of a shopping cart Dynamic Business Object. Receive Customer Request *and* Notify Customer *are operations inherited from the* Abstract Customer Request Module.

The Customer Request workflow is extended to address the new requirements for handling a shopping cart request, as in Figure 6.

By extending the Abstract Customer Request Module, only the *Process Customer Request* operation was replaced with two new ones: *Verify Customer Credit Card* and *Ship Items to Customer*.

This example is not meant to be an exhaustive analysis of the Shopping Cart process, but rather to present a clear process of employing Dynamic Business Object Pattern to address a specific business problem at hand and provide a solution via the proposed pattern.

The Dynamic Business Object Pattern can be further enhanced by applying security patterns to the modules' objects and workflows, as in the Securing Analysis Patterns [2].

4.1 Consequences

4.1.1 Benefits

Dynamic Business Object Pattern can greatly simplify the development of business objects, especially if a growing library of dynamic business objects is accumulated and employed over time. The pattern provides a clear approach for describing a given business object and its contents, even when high complexity is present. The pattern also takes into account the processes (via workflows) that are required to support the business object's life cycle. The pattern is also flexible enough to allow for future changes and additions, if any should be required.

4.1.2 Liabilities

Dynamic Business Object Pattern does not take into account the specifics of dynamic data object support at the user interface and data persistence levels. In addition, until a Dynamic Business Objects library is amassed, much of the work will be required to be performed from scratch.

5. RELATED PATTERNS AND FRAMEWORKS

5.1 The Document-View-Presentation (DVP) Pattern

DVP [8] separates an application into three components: document, view, and presentation. The document component holds business logic and data. The view component is responsible for service requests and supplying the data to the document. The presentation component processes the events and provides data to the view component.

The Dynamic Business Object Pattern differs from the DVP in that it separates the objects into two types: dynamic data object (DDO) and static document (SD). Unlike DVP, Dynamic Business Object Pattern leaves the retrieval and presentation of data to the DDO. In addition, the presentation of SD is left to the application handling the specific type of document (i.e. MS Word for .doc and .docx types, Adobe PDF for .pdf type, etc). Finally, Dynamic Business Object Pattern offers a workflow component, which is not available in DVP.

5.2 The Model-View-Controller (MVC) Pattern

MVC [9] has become one of the most popular and widely accepted patterns in the market today. Its main benefit is that it separates the presentation of information from the business logic and data storage. The model component in MVC is responsible for business logic and data storage. The view component handles the end user presentation of the requested data. The controller component handles the input from the view component and converts it into instructions for the view and/or model. The Dynamic Business Object Pattern differs from MVC in that data retrieval and presentation are performed by the dynamic data object in Dynamic Business Object Pattern, and that static document object presentation mechanism is left to the external application responsible for handling the type of static document requested. Unlike Dynamic Business Object Pattern, MVC can only support a request/response mode, and does not provide the means for handling a business process in the form of a workflow.

5.3 The Presentation-Abstraction-Control (PAC) Pattern

PAC [10] defines a structure for interactive software systems in the form of a hierarchy of cooperating agents. In PAC, every agent is responsible for a specific area of the application's functionality and is comprised of three components: presentation, abstraction, and control. The abstraction component in PAC is similar to MVC's model component. The presentation component in PAC can be viewed as a combination of view and controller components in MVC pattern. The control component is responsible for facilitation between PAC agents.

Dynamic Business Object is similar to PAC's agent in that it can define a single business object, or a group of business objects comprising a module (or a system in PAC). In PAC, agents are responsible for data retrieval, presentation and maintenance. Dynamic Business Object Pattern differs here from PAC in that it classifies the data as dynamic or static, and consequently uses different presentation mechanisms for each. In addition, the job of business processes is handled by the workflow component in Dynamic Business Object Pattern, whereas in PAC it is the responsibility of the agent itself.

5.4 The Common Business Objects Framework (CBOF)

The Common Business Objects Framework was proposed by the Object Management Group (OMG). It is based on OMG's CORBA and Business Objects Framework specification, which handles business concepts, processes and events. Like Dynamic Business Object Pattern, the Common Business Objects in CBOF represent the business rules, functions and processes. Also like Dynamic Business Object Pattern, CBOF groups CBOs into different groups (modules in Dynamic Business Object Pattern), each group specializing in particular business area (i.e. Finance, Health Care, Manufacturing, etc.). Unlike CBOF, the Dynamic Business Object Pattern is not bound by any specific technology. CBOF is a particular framework technology, tightly coupled with the Common Object Request Broker Architecture (CORBA) and relies on many of its feature to make the Common Business Objects work.

5.5 Other Business Object Frameworks

In the 90's the Common Object Request Broker Architecture (CORBA) [11] was hailed by the Object Management Group (OMG) as the ideal solution for Business Objects landscape. CORBA was not widely accepted and was survived by Java 2 Enterprise Edition (J2EE) [19] and Microsoft .NET [20] platforms. Today most of the commercial business applications are built on J2EE and .NET frameworks. In recent years the Spring framework [21] has shown a tremendous popularity as the alternative to J2EE, in large part due to its simplicity as compared to J2EE, and because it also uses Java as the main programming language. Last, but no least are the business applications build by SAP AG [22]. Considered to be the largest developer of business applications in the world, commanding 25% of the market [23], SAP uses its own proprietary framework comprising of ABAP language and BAPI (Business API) interfaces. Unable to resist the tremendous growth of Java language, SAP has also introduced a Java based framework it calls Netweaver, which is increasingly used to build SAP's web based commercial applications.

There are also several commercial products in the area of Business Objects presentation using XML. Among them are SmartClient [13], Adobe Flex [14], Sencha GXT [16], Vaadin [17] and Icefaces [18] software products .

5.6 Web Business Objects

In recent years the Service Oriented Architecture (SOA) has gain a great amount of popularity, and many business applications have been developed using the SOA. The main attraction of SOA over other frameworks is that, with exception of XML, it is language independent, and is entirely web based. A great example of SOA business application is federal government's Federal Procurement Data System – Next Generation [12], one of the earlier adapters of SOA.

6. CONCLUSIONS AND FUTURE WORK

The Dynamic Business Object Pattern provides a reusable and flexible approach to address just about any business problem and offer a solution based on proven methods. Furthermore, by isolating different business functions into specific modules, we can narrow our analysis to specific modules, and employ that module's pattern to solve the business problem at hand.

Further research is needed to see how the Dynamic Business Object Pattern would work with more complex models, those including tens or hundreds of objects and workflows. Also of great interest is the transition to design/implementation model, as well as opportunities for code generation. The available storage options and how they can be incorporated into the pattern are also of significant interest, especially when taking into account the recent advances in and popularity of NoSql databases. Finally, the delivery and presentation of dynamic business objects to the end users from a platform-neutral specification is of great interest as the wide range of the devices used by businesses has grown exponentially over the years. The availability of business applications is no longer expected to be on desktops and laptops only. New avenues for using the Dynamic Business Object Pattern should be explored for pervasive computing and mobile peer-to-peer computing applications.

7. ACKNOWLEDGMENTS

We would like to thank Prof. David West for his valuable feedback and support during the shepherding of this paper.

8. REFERENCES

[1] Fernandez E.B. and Yuan X., 2000. Semantic Analysis Patterns, Department of Computer Science and Engineering Florida Atlantic University.

[2] Fernandez E.B. and Yuan X., 2007. Securing Analysis Patterns, Department of Computer Science and Engineering Florida Atlantic University.

[3] Gamma E., Helm R., Johnson R., Vlissides J., 1994. Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley.

[4] Fowler M., 1997. Analysis Patterns: Reusable Object Models, Addison-Wesley Longman.

[5] Daum B., Modeling Business Objecs with XML Schema, 2003 Morgan Kaufmann Publishers.

[6] Sridaran R., Padmavathi G., Iyakutti K., A Survey of Design Pattern Based Web Applications.

[7] Buckl S., Matthes F., Monahov I., Roth S., Schulz C., Schweda C.M., 2012. Enterprise Architecture Management Patterns for Company-wide Access Views on Business Objects, ACM Transactions on Applied Perception, Vol. 2, No. 3, Article 1, Publication date: May 2012.

[8] Chang K.Y., Chen L.S., Lai C.K., 1999. Document-View-Presentation Pattern, Department of Electrical Engineering, National Cheng-Kung University, Taiwan.

[9] Krasner G.E. and Pope S.T., 1988. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, Volume 1 Issue 3 (Aug./Sept. 1988), 26-49.

[10] Buschmann, F., Meunier R., Rohnert H., Sommerlad P., Stal M., 1996. Pattern-Oriented Software Architecture: A System Of Patterns. West Sussex, England: John Wiley & Sons Ltd.

[11] Business Object DTF, Common Business Objects, 1997. OMG Document bom/97-12-04.

[12] FPDS-NG, Federal Procurement Data System - Next Generation, 2013. DOI=https://www.fpdsng.com.

[13] SmartClient Enterprise Edition, Isomorphic Software, 2010. DOI=http://smartclient.com/product/smartclient.jsp.

[14] Adobe Flex, Adobe Systems Incorporated, 2013. DOI=http://www.adobe.com/products/flex.html?promoid=DINEZ.

[15] Ariba Inc., 2013. DOI=http://www.ariba.com

[16] Sencha GTX, Sencha Inc., 2013. DOI=http://www.sencha.com/products/gxt.

[17] Vaadin UI Components, Vaadin Server & Client, Vaadin Ltd., 2012. DOI=https://vaadin.com/home.

[18] Icefaces, ICEsoft Technologies Inc., 2013. DOI=http://www.icesoft.org/java/home.jsf.

[19] J2EE, Oracle Corporation, 2013. DOI=http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[20] .NET, Microsoft, 2013. DOI=http://www.microsoft.com/net.

[21] Spring Framework, GoPivotal, Inc., 2013, DOI=http://www.springsource.org/spring-framework.

[22] ABAP and BAPI, SAP AG, 2013. DOI=http://scn.sap.com/community/abap.

[23] Columbuss, Louis, May 12, 2013. 2013 ERP Market Share Update: SAP Solidifies Market Leadership. Forbes.

[24] Wong K. and Bass D., May 23, 2012. SAP to Acquire Ariba for $4.3 Billion in Push Into Cloud. Bloomberg.