SMART BUSINESS OBJECT
A New Approach to Model Business Objects for Web Applications

Xufeng (Danny) Liang
School of Computing and Mathematics, University of Western Sydney, Sydney, Australia

Athula Ginige
School of Computing and Mathematics, University of Western Sydney, Sydney, Australia

Keywords: Business Object, Modelling Language, Web Engineering, Rapid Development.

Abstract: At present, there is a growing need to accelerate the development of web applications and to support continuous evolution of web applications due to evolving business needs. The object persistence capability and web interface generation capability in contemporary MVC (Model View Controller) web application development frameworks and model-to-code generation capability in Model-Driven Development tools has simplified the modelling of business objects for developing web applications. However, there is still a mismatch between the current technologies and the essential support for high-level, semantic-rich modelling of web-ready business objects for rapid development of modern web applications. Therefore, we propose a novel concept called Smart Business Object (SBO) to solve the above-mentioned problem. In essence, SBOs are web-ready business objects. SBOs have high-level, web-oriented attributes such as email, URL, video, image, document, etc. This allows SBO to be modelled at a higher-level of abstraction than traditional modelling approaches. A lightweight, near-English modelling language called SBOML (Smart Business Object Modelling Language) is proposed to model SBOs. We have created a toolkit to streamline the creation (modelling) and consumption (execution) of SBOs. With these tools, we are able to build fully functional web applications in a very short time without any coding.

1 INTRODUCTION

Web programming languages (such as PHP, Python, Perl, ASP, Java, etc) and database technologies have been around for a long time with major web applications developed using them. However, with the increased time-to-market pressure, we can no longer afford the time to work with rows and columns in sophisticated databases, and create business web-based applications from scratch. Thus, there is a growing need to rapidly develop web applications that can evolve meeting the ever-changing business needs. One of the challenges in developing web applications is to minimise the gap between the development domain and the actual problem domain. This has led to investigate ways of creating better modelling techniques that empowers users to express their mental model at a higher-level of abstraction. Further to find smarter tools that can capture and convert, for implementation, those models into software objects in order to create powerful web applications (i.e. by executing those models). Our work builds on early work done by Reenskaug in MVC (Model-View-Controller): a modelling approach to bridge the gap between users' mind and computer data (Reenskaug, 1979b, Reenskaug, 1979a). Moreover, empowering users and allowing trained end users to maintain or even enhance existing applications is a cost-effective way to support web application evolution (Wulf and Jarke, 2004).

The OO (Object-Oriented) paradigm provides us with techniques to build software applications by mapping real world objects directly into software objects. In the past, object mapping techniques have proven to be successful in software engineering projects (Casey, 1999). These techniques provided a natural correlation between real world objects and objects in the software and database domain. Additionally, OO design techniques are applicable

(Danny) Liang X. and Ginige A. (2006).
SMART BUSINESS OBJECT - A New Approach to Model Business Objects for Web Applications.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 30-39
DOI: 10.5220/0001317800300039

of handling the domain evolution (Barstow and Arango, 1991). The encapsulation concept in OO provides us with a systematic way to handle software evolution. System behaviours are encapsulated inside the objects as methods. This provides a means for software evolution to be handled gracefully by delegating responsibilities to objects. The ability to systematically handle software evolution makes object orientation a suitable technique for implementing web applications.

However, the traditional object concept has a low level of abstraction and has been designed for use by software developers. On the contrary, business objects are "business-focused" software objects modelled to represent real world business entities (Lhotka, 2003). They operate at a higher level of abstraction than software objects. Business objects offer representations of organisational concepts, such as resources and actors, which collaborate with one another in order to achieve business goals (Caetano et al., 2005). Maamar and Sutherland (2002) state that business objects provide "an insight into what aspects of a business should be delegated, how these aspects may evolve, and what will be the effect of specific changes", and through business objects, "managers and users can understand each other by using familiar concepts and creating a common model for interactions". Thus, an important attribute that distinguishes business objects from traditional software objects is the fact that they can be understood by both business (business managers and users) and software (software developers and the software itself). They are considered as the bridge between software developers and domain experts.

While object-orientation has been long proven suitable for building business applications, existing web development tools and frameworks do not accommodate the need for high-level modelling and rapid development of web-based business applications. What is required instead are business objects that make provision for web interfaces and behaviours, we call those web-ready business objects. Web-ready business objects should have associated conventions such as:

- Providing a file upload facilities for documents or other binary media contents
- Displaying URL as hypertext links, emails addresses as mailto hypertext links
- Rendering calendars to assist user to enter date information
- Offering the suitable media players for video content.

In order to speed up the development of web-based business applications, we need business objects that incorporate those conventions. These conventions are imperative directives that contribute to "web-readiness" of business objects.

As a consequence, web-ready business objects should embrace semantic-rich, web-oriented attributes. These web-oriented attributes have high-level, semantic-rich abstract data types (ADT) such as: email, URL, image, video, document, and date. These abstract data types require special validation logic, content handling methods, and presentation mechanisms. An image attribute for example, we need to validate its filename appropriately, provide an upload facility to record the image's filename to the database and store the actual image file to a preconfigured location on the server (assuming that we are not storing binary data inside the database), and render the file content as image to the web browser (via the `<img>` tag if HTML is used). Web-oriented attributes can affect different layers of a web application. The benefit of being able to program using abstract data types is well understood in programming (see (Liskov and Zilles, 1974)). Over decades, programmers have taken advantage of language-provided, built-in data types, such as "integer", to perform normal operations, such as arithmetic calculations, without worrying about the underlying low-level instructions that are required to be carried out by the machine. Similarly, in the context of web applications, we need the direct support for using richer and higher-level abstract data types in order to represent web-oriented attributes of business objects.

At present the responsibility of handling these web-oriented richer data types is passed down to the applications logic, based on primitive data types, such as "string" or "text". For example, an email address attribute is not considered as type "email", but type "string" or "text". As a consequence, web developers need to craft the same regular expression for validating the email address from users' input and customise the necessary web templates to render the email attribute as an email hypertext link (mailto) in every web application they build. This is mainly due to the fact that "email" is not a built-in, language-provided data type. The missing notion of web-ready business objects does not only decelerate web application developments, but also poses impediments to business objects being modelled at a higher level of abstraction. We cannot simply model: "Employee has photo", and expect a file upload facility is provided for updating the photo

attribute and a correctly displayed image of the uploaded photo is rendered for viewing.

Thus, in this paper, we propose a Smart Business Object (SBO) concept. SBO is designed to empower users by addressing the issues in modelling and building web applications. SBO uses representations of business objects and their attributes to achieve a higher-level of design abstraction in web applications leading to faster development. We will demonstrate the concept of SBO through the use of the lightweight SBO Modelling Language (SBOML) to model a SBO and create different views of SBO as web applications.

2 RELATED WORK

Recent MVC web development frameworks such as (Apple, 2001, Catalyst, 2005, Ruby on Rails, 2005) and Model-Driven Development tools such as (AndroMDA, 2005, openMDX, 2005, Tangible Engineering, 2005) provided the capability of auto generating basic web user interfaces for CRUD (Create Retrieve Update Delete) operations for user-defined persistent objects. The built-in capabilities of object persistence and web presentation UI generation in contemporary tools or frameworks have simplified the process of developing business objects for the web. However, the real-world semantics and the high-level abstraction required for rapid modelling and developing of business objects for enterprise web applications is still missing.

Most current tools rely on low-level database column types to determine the web presentation UI for the corresponding business object attributes. For example, an attribute is rendered as a textbox if its column type in the database is 'text'. However, the semantics offered by database column type is insufficient for defining business objects in web applications. It is tedious and unproductive for developers having to craft the same regular expression to validate the email attribute of a business object. For example, in Ruby on Rails, each time we need to validate an email address attribute of a business object, we need to code the same regular expression:

```
class Employee < ActiveRecord::Base

    validates_presence_of :first_name,
:last_name, :email
    validates_format_of :email,:with =>
/^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-
z]{2,})$/
end
```

Modelling business objects in most model-driven tools via UML class diagram variants are also low-level and lack high-level semantics suitable for modelling web-ready business objects. For example, we have to model "Employee has email:string" and then customise in different layers of an web application (such as presentation layer and domain layer (refers to the layers of enterprise application defined in (Fowler, 2002))) in order to make the email attribute to be rendered as mailto hypertext link and to be validated properly from user inputs.

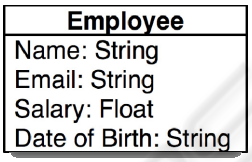

Figure 1: An UML class diagram for an employee class.

The Naked Object (Pawson and Matthews, 2002) address the problem of "behaviour completeness" in business objects. Approaches such as ARANEUS (Atzeni et al., 1998), WebML (Ceri et al., 2000), and OOHDM (Rossi et al., 2000) have focused on issues surrounding the modelling of content, navigation, and structure in web applications. Most of them have abstracted content into traditional business objects. However, none has looked at "web-readiness" of business objects and theirs potential in raising the level of abstraction in modelling business object in order to accelerate the development of web applications.

3 SMART BUSINESS OBJECT

Smart Business Object (SBO) is a web-ready business object that supports semantic-rich, web-oriented attributes suitable for implementing web-based business applications. As previously mentioned, these attributes will support convention settings such as file upload facility for documents, displaying URLs as hypertext links, rendering a calendar to assist user to enter date information, etc. This enables SBOs to auto generate appropriate web interfaces that will accelerate the development of web-based business applications. Figure 2 is an example of rendering a class of SBO called "employee" as a web table with search capability. In this example, the contents of the email address attribute are displayed as mailto hypertext links and the content of the photo attribute is displayed as images.

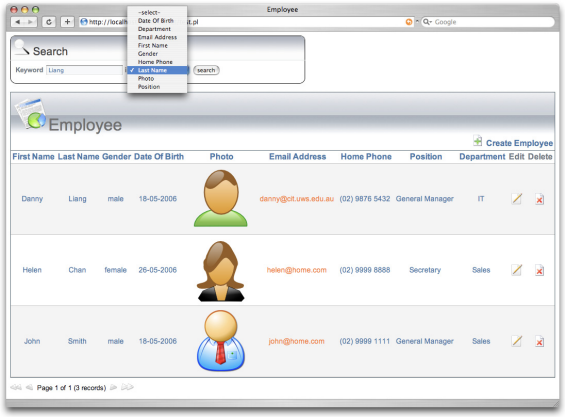

Figure 2: Rendering an SBO as a table with search capability.

To assist the generation of useful web user interfaces (such as in Figure 2) for rapid web application development, each SBO have a rich set of built-in methods (operations) for rendering commonly used web user interfaces, such as tables, forms, navigation menus, etc. These user interfaces allow end users to interact with SBOs via a web browser to perform CRUD operations or execute various custom methods of SBOs.

The one line of code in Perl used to generate the user interface in Figure 2 is as follow:

```
organisation::employee-
>render_as_search_form(create => 1,
edit => 1, view =>1, delete => 1);
```

In other words, render the "employee" SBO in the "organisation" namespace as a search form, and allow user to have create, view, edit, and delete access to the "employee" SBO.

Users can create fully functional web applications by modelling SBO and executing them to generate various web user interfaces. The concept of web-oriented attributes allows SBOs to be modelled at a high-level of abstraction than conventional modelling approaches.

3.1 High-Level Architecture of Smart Business Object

SBO is a lightweight component that can be easily integrated into existing web frameworks for building both data intensive and process intensive web applications. The SBO is layered on top of a persistent object layer (Figure 3). A persistent object layer is usually realised using ORM (Object Relational Mapping) technologies unless an OODBMS is used. The reference implementation of SBO uses ORM technologies and relational database to achieve object persistence. The Builder component is mainly responsible for modelling SBO. The interpreter for the SBOML lives inside the Builder component.

SBOs are organised by their namespaces. The relationships among SBOs are handled at the object level (as opposed to being at the database level). The advantage of this is that SBOs can establish relationships with other SBOs coming from physically diverse databases. Thus, the role of the Metaobject component is to maintain the relationship definition between SBOs. Moreover, custom SBO schemas can be used to control the behaviours of (e.g. look and feel, localisation, etc) individual SBOs. Thus we need to preserve the mapping information between customs schema and SBOs. This information is also maintained by the Metaobject component. Furthermore, the Metaobject component also maintains the credential information required to connect to the underlying data sources.

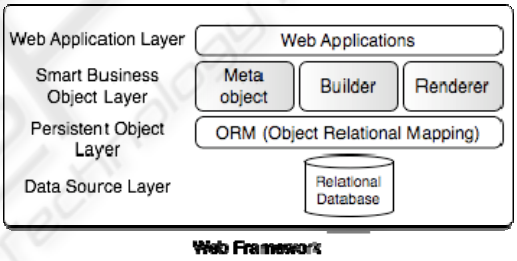

Figure 3: Smart Business Object high-level architecture.

As previously mentioned, SBO have a rich set of built-in methods (operations) for rendering commonly used web user interfaces. The Renderer component is responsible for rendering SBOs. It has a host of APIs (Application Programming Interfaces) to support the generation of various web user interfaces for SBOs. Each API provides a rich set of options to achieve fine grain control over the behaviours of the generated user interfaces. For example, in Figure 2, we have enabled create, edit, view, and delete access for the "employee" SBO. Each API utilises one or more templates. Thus, by specifying customised templates to the rendering APIs, we are able to achieve different look and feel for the generated user interfaces. If the default set of user interfaces are insufficient for certain application, we could extend the existing APIs (by subclassing them) or add new APIs.

3.2 The Smart Business Object Schema

The SBO has a default schema to control the global behaviours of all SBOs. The schema defines a set of default templates used by each rendering API. Thus, by specifying different templates in the default schema or specifying custom schemas, we can change the look and feel of various user interfaces generated by the Renderer.

Additionally, the schema defines the behaviours of each attribute type. By customising the attribute type definitions in the default schema or by specifying custom schemas, we can easily change the behaviours of existing SBO attributes or add new ones.

In turn, each attribute type definition defines a number of behaviours of an attribute, such as: validations, localisation, option values, and formatting and conversion of values. SBO generates the appropriate web user interfaces for its attributes based on their nominated attribute type given when the SBO was modelled. For SBO attributes whose attribute type is not explicitly defined during the time when the SBO is modelled, SBO will aggregate the meta-information of the underlying data source, such as the table definition of a database, and match them against the known attribute types defined in the specified schema to logically derive the most suitable (conventional) web user interfaces for those SBO attributes at run-time.

In this way, the modelling of SBO can be greatly simplified. For example, we can simply model "employee has email", then the generated "employee" SBO automatically and smartly considers its email attribute as being of the high-level type "email" without extra declaration. This feature adds smartness to SBOs. Thus, we are able to achieve a much higher level of abstraction than traditional modelling approaches.

A partial extract from the default SBO schema implemented in XML is given below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<sbo version = '0.0.28'>
    …
    <smartness>1</smartness>
  <table_template>
    table.tt
  </table_template>
    …
    <attribute_definition>
        …
        <attribute>
            <name>salary<name>
            <validate>MONEY</validate>
            <format>
                <to_ui>
            Renderer::_to_ui_money
          </to_ui>
            </format>
            <sort>NUM</sort>
            <default>0.00</default>
            <maxlength>14</maxlength>
        </attribute>
        <attribute>
            <name>photo<name>
            <type>file<type>

    <validate>FILENAME</validate>
            <sort>NAME</sort>
            <convert>
                <to_ui>
            Renderer::_to_ui_file
          </to_ui>
                <to_db>
            Renderer::_to_db_file
          </to_db>
            </convert>
            <format>
                <to_ui>
            Renderer::_to_ui_image
          </to_ui>
            </format>
        </attribute>
        <attribute>
            <name>gender<name>
            <options>
                <option>male</option>
                <option>female</option>
            <options>
        </attribute>
     …
    </attribute_definition>
  …
</sbo>
```

In the example schema, "`<smartness>`" defines whether SBO should automatically derive the high-level attribute types for its attributes. The "`<table_template>`" element defines that the "`table.tt`" template file will be used for the rendering API(s) responsible for rendering SBOs as a web table.

Different attribute types are defined within the "`<attribute_definition>`" element. For example, in the "`gender`" attribute, we have specified two option values: "`male`" and "`female`". For the "`photo`" attribute, we have specified various trigger functions to control the conversion and formatting behaviours. Firstly, we define that "`photo`" is a "`file`" type, such that a file upload input field is provided by default. Before saving the value from users' input (usually via web forms

generate by the SBO), the "`_to_db_file`" function is triggered, such that the filename of the uploaded image file is saved in the underlying database and the actual binary image file is saved on a preconfigured location on the server. Similarly, during retrieval, each value of the "`photo`" attribute is sent to the "`_to_ui_file`" trigger function in order to construct the necessary URL path needed access the image file on the server. Then it is sent to the "`_to_ui_image`" function for formatting, such that the values are displayed as images on users' web browser (such as via the HTML `<img>` tag). The "employee" SBO in Figure 2 is rendered utilising various high-level attribute types defined in the default SBO schema, including "`photo`" attribute type that we have just discussed. We can always define new trigger function in order to handle special attribute types.

3.3 Smart Business Object Modelling Language

According to Pilone and Pitman(2005), modelling is "a means to capture ideas, relationships, decisions, and requirements in a well-defined notation that can be applied to many different domains". Domain modelling is the building of an object model of the domain that incorporates both behaviour and data (Fowler, 2002). To streamline the modelling and creation of SBOs, we need a higher-level modelling language. SBOML (Smart Business Object Modelling Language) is a lightweight modelling language designed for modelling SBO.

SBOML is not proposed to be another object-oriented programming language or to extent existing OO concepts. Its main intention is to be a lightweight modelling language that leverage on existing, most commonly used (conventional) OO concepts that are suitable for building web based business applications. It brings OO concepts closer to users' mental model. It is designed to allow users to express their domain specific business objects in near natural language syntax.

In this section, we will use the following conventions to represent the formal construction of the SBOML:

- Keyword elements are emphasised in both bold and italic
- Normal style texts represent user-defined elements
- When an element consists of a number of alternatives, the alternatives are separated by a vertical bar ("|")
- Optional elements are indicated by square brackets ("[" and "]")
- An ellipsis ("...") indicates the omission of a section of a statement, typically refers to recursive statements.

The statement for defining SBO attributes, methods, and 'has' relationships between SBOs is as follow:

```
in namespace, business object has
attribute A [([mandatory] [type] [which
could be option a or option b])],
[might have] [many] another business
object [(has attribute B, attribute C,
yet another business object (has
...))]... ,[ use method A (method name
type from location [option is value,… ]
[with attribute A, attribute B,… | with
attribute A as parameter name abc ,
attribute B as parameter name …]),
service B...]
```

The "*in*" clause defines the namespace where the subsequent business object(s) are created within. If the namespace does not exist, a new namespace is created. The "*has*" clause defines the attributes of the intended SBO, or 'has' relationships with another intended SBO. The optional "*use*" clause defines the methods (operations) of the SBO. We first explain the statement by referencing to a simple example:

```
in organisation, employee has first
name, last name, gender, date of birth,
photo, email address, home phone,
position (has title, description)
```

Literally, we have just defined an "employee" SBO and a "position" SBO where "employee" has a "position". When the above statement is executed, and we can directly render the "employee" SBO to the web, such as to generate a web form for adding new employees (creating new "employee" SBO instances).

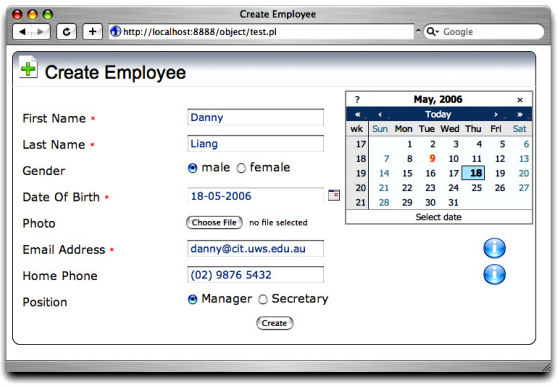

Figure 4: Rendering the "Employee" SBO as a web form.

As previously mentioned, by default, a SBO predicts its attribute types by matching the attribute name against the defined attributes types in the default SBO schema. Thus, when rendered as a web form (Figure 4), the "employee" SBO automatically:

- Enforce first name, last name, date of birth, and email address attributes as mandatory fields and enforce the appropriate validation rules to all corresponding fields
- Provide the "male" and "female" option values to the gender attribute according to SBO schema
- Provide a calendar to assist users for date entry for the "Date of Birth" attribute and present the date according to users' locale setting
- Provide a file upload facility for the photo attribute to upload binary image file
- List the available positions as options items (assuming that we have previously created some "Position" SBO instances), due the relationship established between the "employee" SBO and the "position" SBO

We can always overwrite the default settings, and explicitly declare the attribute type. For example:

```
in organisation, employee has first
name, last name,…, department
(mandatory name which could be IT or
Sales)
```

By default, all defined attributes are optional, except for whose attribute types are defined as mandatory in the specified SBO schema or due to the requirement of the underlying data source (such as a NOT NULL column of a database table). In the example, the "***mandatory***" keyword enforces that the value of the department attribute cannot be empty (i.e. a mandatory field on a web form). The "name" specifies the type of attribute. Thus, could be any attribute type defined in the default SBO schema or in any custom SBO schema. The "***which could be … or***" clause allows users to specify the possible value set of an attribute. In case of the department attribute in the example, option values are "IT" and "Sales".

The "***many***" keyword indicates a "has many" relationship, in UML terms, the cardinality is [1..*]. In combination with the "***might have***" keyword, i.e. "***might have many***", then the cardinality becomes [0..*]. For example:

```
in organisation, employee has first
name, last name,…, might have many
office (has room number, building id)
```

SBO can easily aggregate local functions or remote service as its methods (operations). This enables SBO to be seamlessly integrated with workflow engines and SOA (Service-Oriented Architecture) to develop more complex process oriented business web applications. This can be achieved using the "***use***" clause. For example:

```
in organisation, employee has first
name, last name,…, use notify HR
(notify_HR from
http://10.10.10.2/notify.wsdl with
first name as param_first_name, last
name as param_last_name)
```

In the example, "`notify HR`" is the name of the method for the employee SBO, and "`notify_HR`" is the actual name of the remote method "`from`" the WSDL file located at "`http://10.10.10.2/notify.wsdl`". In the reference current implementation, the SBO support Web Services and XML-RPC for remote invocation. Thus, the "***type***" keyword could be: Local (for executing local application APIs), Web Service, or XML-RPC. The "***with***" keyword is used to indicate the mapping of the attributes of the SBO to the required parameters of the remote method. In the example, when the "`notify_HR`" method is executed, the value of the first name and last name of an "employee" SBO instance is passed to the "`param_first_name`" parameter and the "`param_last_name`" parameter respectively. Depending on the nature of the remote method, more arguments, such as URI, may be required to identify and execute the remote method, thus the clause within the squarely blanket allows user to specify key-value pairs for any optional argument that is needed.

After incorporating the changes to example shown in Figure 4, when we retrieve an instance of the "employee" SBO and render it as a web form again, it would generate the screen shown in Figure 5. Now, users can assign a department, multiple offices to the employee and click the "notify HR" button to execute a web service.

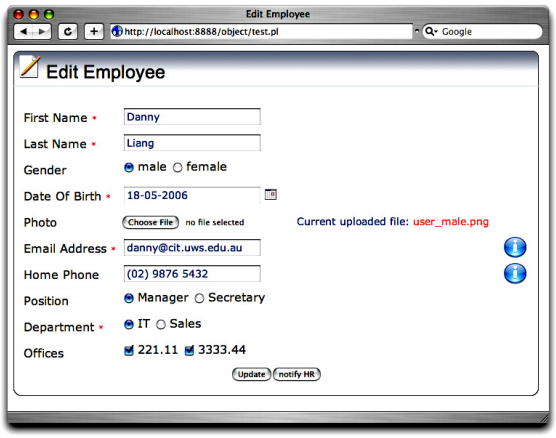

Figure 5: Rendering the "Employee" SBO as a web form with a department, multiple offices and a new method.

We can use the following statements to explicitly define relationships among existing SBOs. For "has" relationships, we could use either:

```
in namespace, business object has|
might have another business object [as
attribute X] [via yet another business
object]
```

or

```
business object in namespace has| might
have another business object in another
namespace [as attribute X] [via yet
another business object in yet another
namespace]
```

The second construct allows SBOs to establish relationships across namespaces. The "as" clause is to nominate a specific attribute as a reference to a foreign SBO instance. For example:

```
in organisation, employee might have
employee as supervisor
```

In the above example, the "employee" SBO has a self-referential "has" relationship, such that an employee may have a supervisor, which is also an employee.
The "via" clause is a shorthand for specifying "many-to-many" relationships. For example:

```
in organisation, employee has car via
company car rental
```

Similarly, to define "is a" relationships (inheritance) between SBOs, we can use either of the following statements:

```
in namespace, child business object is
parent business object
```

or

```
child business object in namespace is
parent business object in another
namespace
```

For example:

```
in organisation, employee is person
```

3.4 Creating Web Applications using Smart Business Object

The reference implementation of SBO is deployed on a web framework called CBEADS© (Ginige et al., 2005). We have created a lightweight SBO toolkit, which consists of the SBO Builder (Figure 6) and the SBO User Interface Generator (Figure 7) on the CBEADS© framework. They are designed to streamline the creation (modelling) and consumption (execution) of SBOs.

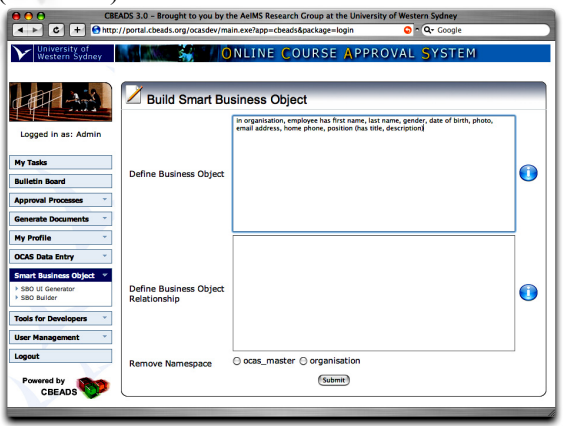

Figure 6: The SBO Builder.

The SBO Builder allows users to model and create SBOs and relationships among them using the SBOML. The SBO User Interface Generator allows users to easily create applications on the CBEADS© framework by rendering SBOs using the SBO rendering APIs. It also allows users to customise

various options supported by the SBO rendering APIs.

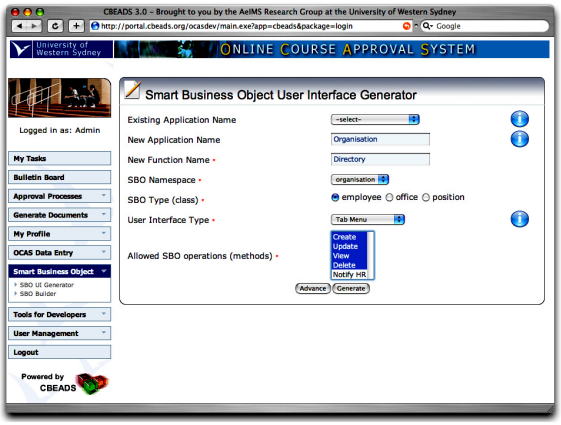

Figure 7: The SBO User Interface Generator.

The SBO toolkit allows fully functional web applications to be created without any coding. Figure 8 is an employee directory application generated purely using the toolkit. It is rendering all the SBOs in a namespace as a tab menu view.

Figure 8: View of the generated application.

3.5 Creating a Customer Relationship Management (CRM) Application

In this section, we will demonstrate how we can use SBOs to generate a lightweight CRM application on the CBEADS© framework. According to the actual business requirements, we need to first identify the actors and their actions, for example:

- Potential customers can make enquiries about products, request sales people to visit them to discuss about products, and make purchases
- Sales persons need to keep track of customers, visits, and sales orders.

Next, we need to identify the necessary business objects:

- A customer has first name last name, email address, phone number, and address
- A sales person has first name, last name, and phone number
- An enquiry has title, question, answer, and date
- A product has code, name, description, price, and enquiries
- A sales order has number, customer, sales person, products, and total amount
- A visit has title, date, time, description, customer, sales person, and sales order.

Using the SBO Builder tool, we can generate those business objects by expressing them in SBOML:

```
in crm, visit has title, date, time,
description, customer (has first name,
last name, email address, phone number,
address), sales person (has first name,
last name, phone number), sales order
(has number(mandatory alphanumeric),
many product (has code, name,
description, price, many enquiry (has
title, question, answer, date)), total
amount)
```

The above SBOML expression models all the identified business objects at the same time. However, we can also model them individually.

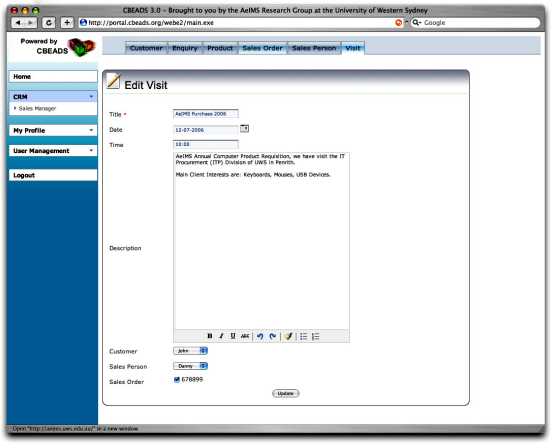

Figure 9: Sales management function assigned to sales people.

Lastly, we generate various views (such as Figure 9) of the SBOs using the SBO User Interface Generator and assigned them to the system user groups defined in CBEADS©. Similarly, we can easily extend the CRM application by modelling additional business

objects, such as suppliers or competitors, to meet the evolving business needs.

Once the business objects and the views of the business objects based on the actions actors need to perform are identified, we can quickly generate fully functional web-based applications using SBO toolkit and the CBEADS© framework. Thus, the overall development time can be greatly reduced.

4 CONCLUSION

In this paper, we have introduced the Smart Business Object concept. The SBOs support semantic-rich, web-oriented attributes. We have presented a modelling language that allows users to express their mental model at a higher-level of abstraction. We have created a tool that generates web-ready Smart Business Objects from the high-level models. We have demonstrated the significant benefits of utilising Smart Business Object in web application development such as ability to model the application based on high level business domain objects and very rapid development of the application using the tools that we have created.

We have implemented several industry projects using SBOs. A significant project is an enterprise level application; the Online Course Approval System (OCAS) (University of Western Sydney, 2006) developed for University of Western Sydney (UWS). The use of SBOs greatly reduced the low level modelling activities such as creating ER diagrams and database schemas and enabled us to rapidly develop OCAS.

REFERENCES

Andromda (2005) Cutting Edge MDSD/MDA Toolkit.

APPLE (2001) WebObjects 5 Reviewer's Guide.

Atzeni, P., Gupta, A. & Sarawagi, S. (1998) Design and maintenance of data-intensive web-sites. *the 6th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'98)*. Springer-Verlag.

Barstow, D. & Arango, G. (1991) Designing software for customization and evolution. *Proceedings of the 6th international workshop on Software specification and design.*

Caetano, A., Silva, A. R. & Tribolet, J. (2005) Using roles and business objects to model and understand business processes. *Symposium on Applied Computing.* Santa Fe, New Mexico, ACM Press.

Casey, R. M. (1999) Object Mappings in a Software Engineering Project. *Software Engineering Notes - ACM SIGSOFT,* 24.

Catalyst (2005) Welcome to Catalyst Development.

Ceri, S., Fraternalli P. & Bongio, A. (2000) Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. *WWW9 Conference.*

Fowler, M. (2002) *Patterns of Enterprise Application 0Architecture*, Addison-Wesley Professional.

Ginige, J. A., Silva, B. D. & Ginige, A. (2005) Towards End User Development of Web Applications for SMEs: A Component Based Approach. *ICWE 2005.* Sydney, Australia.

Lhotka, R. (2003) *Expert One on One Visual Basic .NET Business Objects,* Birmingham, Wrox Press Ltd.

Liskov, B. & Zilles, S. (1974) Programming with Abstract Data Types. *Symposium on Very High Level Programming Languages.*

Maamar, Z. & Sutherland, J. (2002) Toward intelligent business objects. *Communications of the ACM,* 43.

OPENMDX (2005) openMDX - the leading open source MDA platform.

Pawson, R. & Matthews, R. (2002) *Naked Objects*, John Wiley and Sons Ltd.

Pilone, D. & Pitman, N. (2005) *UML 2.0 in a Nutshell,* Sebastopol, O'Reilly Media, Inc.

Reenskaug, T. (1979a) MODELS - VIEWS - CONTROLLERS.

Reenskaug, T. (1979b) THING-MODEL-VIEW-EDITOR:an Example from a planning system.

Rossi, G., Garrido, A. & Schwabe, D. (2000) Navigating between objects. Lessons from an object-oriented framework. *ACM Computing Surveys (CSUR),* 32.

Ruby on Rails (2005) Web development that doesn't hurt. Ruby on Rails.

Tangible Engineering (2005) Tangible Architecture.

University of Western Sydney (2006) Online Course Approval System (OCAS). University of Western Sydney (UWS).

Wulf, V. & Jarke, M. (2004) The Economics of End-User Development. *Communications of ACM,* 47.