Naked Objects Framework

Aruna Raja
Usha Mittal Institute of Technology,
SNDT Women's University, Santacruz
(W), Mumbai - 400 049

Devika Lakshmanan
Usha Mittal Institute of Technology,
SNDT Women's University, Santacruz
(W), Mumbai - 400 049

ABSTRACT
With more and more business systems designed using object oriented approach, written using object oriented languages like Java and interfaces created through various tools, there is need for such systems to show the quintessence of object orientation, which is 'behavioural completeness'. This paper discusses Naked Objects Framework (NOF), which is an open source prototyping tool based on Java that auto generates an entire system that is behaviourally complete by plainly defining behaviourally rich objects that can be easily accessed by the user. The resulting object oriented interface is very user friendly and allows both the developer and the client to equally contribute in the requirements specification phase.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – elicitation methods, tools

D.2.2 [**Software Engineering**]: Design Tools and techniques – Object-oriented methods

H.5.2 [**User Interfaces**]: Prototyping

General Terms

Performance, Design

Keywords

Naked Objects Framework, behavioral completeness, prototyping, object oriented user interface, drag and drop, web view, domain objects, actions, services, repositories, agility.

1. INTRODUCTION
Naked Objects is an open source application development platform for Java. The Naked Objects Framework is used to create business systems from behaviorally complete business objects. Initially created as a prototyping tool for user interface, it is now used as for prototyping the entire object model with the inclusion of persistence, sharing and distribution of objects.

It is called 'Naked Objects' because all that is to be created are the domain models as 'Plain Old Java Objects' (POJO).Two things have made this possible. The first is that there is no application-specific user interface code - the UI for both systems is created 100% automatically from the business object definitions. The second is that the Naked Objects approach encourages very good object-oriented design, specifically, very high levels of polymorphism.

Following are the three principles of Naked Objects pattern:

1. All business logic should be encapsulated onto the domain objects.
2. The user interface must completely reflect the domain objects including all user actions such as creating and retrieving domain objects.
3. The user interface creation must be entirely automated from domain objects.

The main focus has been on creating agile systems so that they support changing business requirements resulting in good levels of reusability and thereby reducing development time.

The visual appeal of the interface supports even a non-IT user to be involved in the design specification phase. The feature of authorization control ensures that an entire organization can make use of the same system hence supporting uniformity throughout.

2. OBJECT ORIENTATION IN NAKED OBJECTS FRAMEWORK

The Object Oriented approach towards development of software systems has been around for a long time. It has gained immense popularity because every object mirrors a real world entity thus helping the programmer to visualize the system very effortlessly. Many business systems designed using an object – oriented methodology support features such as encapsulation and inheritance but miss out on a very important concept of 'behavioural completeness'. It means that every software object must not only represent the properties of the corresponding real – world entity but also implement their behaviours. This does not imply that all possible behaviours must be modelled but the data and procedure should not be separated i.e. they should only be properties of the object and not implemented elsewhere in the system.

Naked Objects Framework supports this principle through behaviourally rich objects that reflect real – world business domain. Each object is represented in the form of an icon and behaviours as actions on pop – up menus. Thus NOF captures the central idea of an Object Oriented User Interface and at the same time allows the objects to be exposed directly to the user rather than hiding them under a UI. The user can directly operate upon the business objects or their classes without the interference of

any menus, scripts or dialog boxes making it user friendly and hassle free.

3. MULTIPLE VERSIONS

3.1 Drag and Drop

Each domain object is represented in the form of an icon and methods as actions on pop-up menu. Since it makes heavy use of Drag and Drop, it is labeled as 'DND' user interface as shown in figure 1.

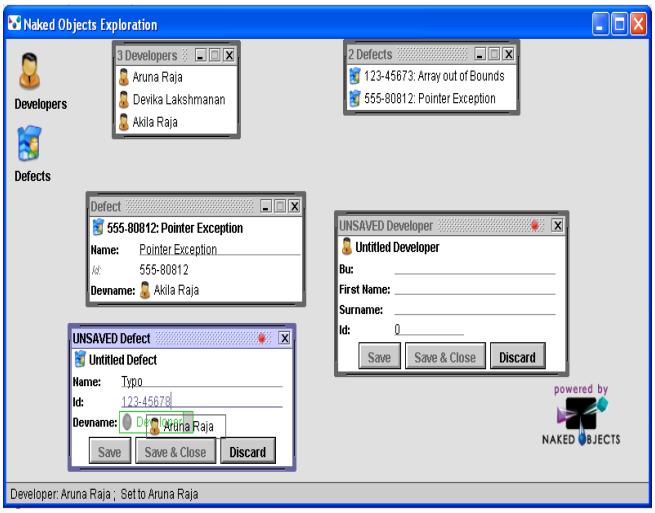

Figure 1. DND Version

3.2 Web View

Web view is implanted on a web browser seen in figure 2. The object's properties are on the right with the action menu on the left. Unlike DND where we drag and drop objects, here drop down menu is used to select objects from a range of valid options. The Web View look can be easily customized through Cascading Style Sheets though it isn't as expressive as Drag and Drop

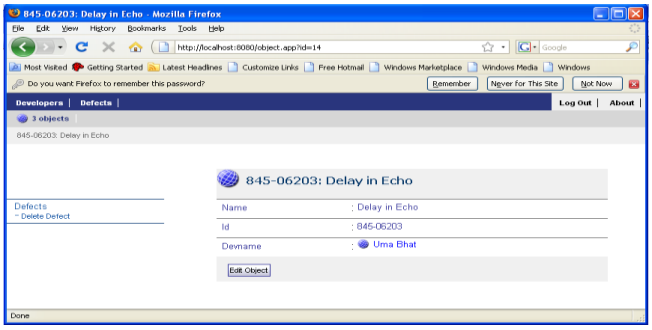

Figure 2. Web View Version

4. CONCEPTUAL DIAGRAM

The entire NOF can be summarized by the conceptual diagram shown below:

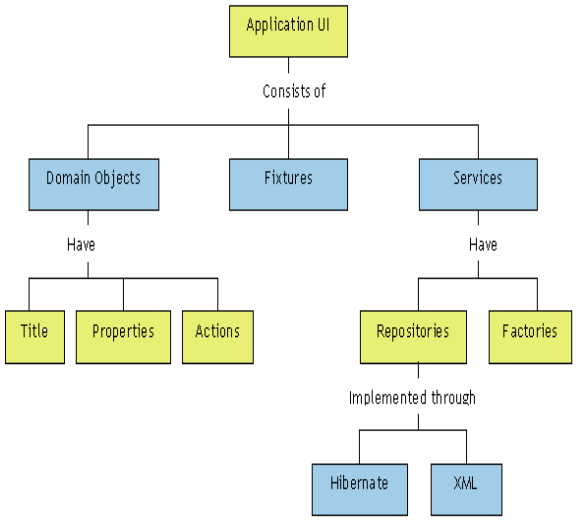

Figure 3. Conceptual Diagram of NOF

The various components of NOF as shown in figure 1 are as follows:

Domain Objects- Behaviourally rich business objects that are represented as icons on the screen. Double clicking on an icon opens a detailed view of its corresponding attributes and relations. Right clicking produces a pop-up menu with actions that can be invoked on the object.

Title- Name associated with each instance of Domain object that appears next to the icon. It helps the user to identify objects and retrieve objects from object store.

Properties- Parameters of the Domain objects in the form of fields such as name, id etc.

Actions- Behaviours of the business objects that are represented in the form of pop-up menus.

Fixtures- Contains data to be launched each time the application is run during demos.

Services- It defines functions that cannot be applied to single instance. E.g.: Find developer, Show all etc. They are defined independently so that their implementation can be changed without affecting the objects that use the services.

Repositories- Complex services required to be defined by user. These are implemented through object stores like Hibernate and XML.

Factories- Services like object creation, which involve common steps is built in Naked Objects Framework through factories.

5. WORKING WITH NAKED OBJECTS FRAMEWORK

Being an approach to domain driven design, to create application using NOF the programmer needs to identify and define the basic artifacts like the Objects, Services, and Factories etc that define the Domain model under consideration.

A Defect Tracking System has been created to understand the implementation process with more ease.

5.1 Defect Tracking System

This business system must manage the various defects logged by the QA team of the organization. It should allow the user to register any new Defects identified. So it should also let user to include any new developer into the system.

5.2 Implementation Details

Identifying and defining the basic artifacts for the above system:

5.2.1 Domain Objects

Developer and Defect can be identified as the main domain objects for the system defined. The Developer object should allow the user to include any new developer into the system and conduct various operations like finding developers by name or id or also delete them from the system if required. The Defect object helps to register any new defects identified. One of the compulsory fields while registering a defect is to allocate a developer from the set of developers in the system.

A separate java source file needs to be created for each object describing their properties and actions. These Domain objects don't inherit from any other objects. However it is quiet useful to extend them from AbstractDomainObject as it helps in reducing several lines of code.

Figure 4. Domain Objects

5.2.1.1Developer Domain Object

The title associated with this domain object is obtained by the combination of First and Surname entered by the user and appears next to the icon.

The parameters for the object includes Business unit (BU), First name, Surname, and a unique Id associated with the developer. Also once added the id of a particular developer can't be changed. BU is the unit of the organization to which the developer belongs. Since abbreviations of BU are generally in use the field has been set such that it only accepts capital letters. The first name and surname fields have a max length set and are allowed to accept alphabets alone. The id field accepts an id entered only if it is unique and with a max length of 6 digits. Also once persisted this field can't be changed.

5.2.1.2Defect Domain Object

The title associated with this domain object is obtained from the name given to the defect.

The parameters for the object include Defect Name, Defect Id and the Developer Name who is assigned the Defect. Defect Name can be an alphanumeric name with white spaces allowed and a max length of 15. The constraint placed on it states that the name should never start directly by digits. Defect Id is a unique id given to the defect with a must follow pattern 'nnn-nnnnn' (where n stands for whole number digits) for uniformity in the Business system. Like in Developers, here too the id can't be changed once the Defect is persisted. In the Developer name field the developer is simply dragged and dropped in and thus there is no need to setup constraints for the same.

Thus the properties and actions for each domain object was defined in separate java source files namely: Developer.java and Defect.java

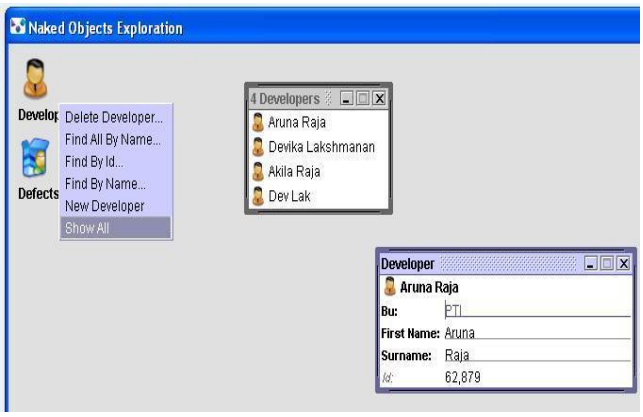

Figure 5. Developer Object's Properties

5.2.2 Services

Services for each Domain object needs to be specified in separate java source file.

Some of the services made available for Developer object include:

showAll - which lists all the Developers present in the system.

findAllByName - which lists all the developers that have the same characters in the last name as entered in the search string.

findByName - lists out the best matching developer that have the same characters in the last name as entered in the search string.

findById - makes use of the unique id linked with each developer to search for his/her existence.

newDeveloper - can be used by the user to create a new instance of the domain object and it provides an expanding form with fields for entering the value for parameters.

deleteDeveloper - helps to delete a persistent developer from the objectstore used.

Some of the services made available for Defect object include:

showAll - It lists all the defects registered in the system.

findById – This returns the defect based on the unique Id entered.

newDefect – Helps the user to register a new defect.

deleteDefect – Helps to delete a persistent defect.

Thus, the above mentioned services were defined for the objects in the DeveloperRepository.java and DefectRepository.java files.

Figure 6. Services defined for Developer Object

5.2.3 Factories

Prototyping applications generally run in the non-persistent mode it is beneficial to define fixtures. Fixtures specify data for the domain objects that is uploaded each time the application is run.

Two java source files namely DevelopersFixture and DefectsFixture were written for the application. Both classes were extended from AbstractFixture class thereby leading to reduction in lines of code.

For Developer object sample data like

```
newDeveloper("PTI","Aruna", "Raja", 62879)
```

and for Defect object sample data like

```
newDefect("Array out of Bounds", "355-40311",getDeveloperRepository().findByName("Lakshmanan"))
```

were uploaded.

5.2.4 Persistence

Its required to use different persistor incase persistence is desired between each run of the application. There are a couple of implementations available:

XML-persistor plugin – This can be used to store objects in a series of XML files. This however isn't suitable for production use, but at least the data doesn't disappear between runs. To use it, just add xml-persistor to the pom.xml and use --persistor xml on the command line.
Hibernate plugin – This can be used to store data in RDBMS.
JPA object store – It also persists to an RDBMS using JPA annotations.

For the defect tracking system persistence was applied using xml persistor.

Additionally, security can be added to the application using authentication and authorization. Authentication can be specified using a passwords file under a config directory. The username : password pair is mentioned in the format specified thereby allowing only valid entries to access the application.

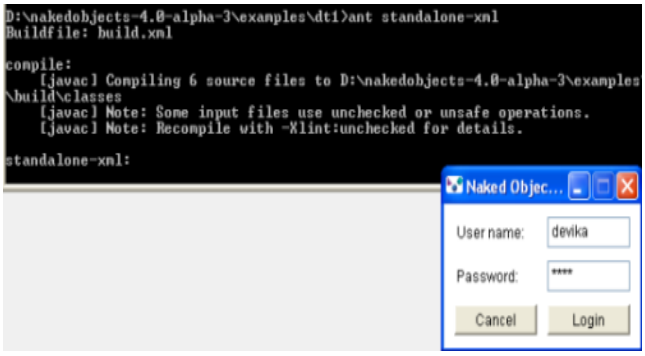

Figure 7. Authentication

Also, Allow and/or Disallow files can be used for authorization to control views as to decide who will get to see what. For this, roles need to be specified additionally with the username : password pair in the passwords file. Allow/Disallow files can be used to clearly allow or disallow access to certain actions or services in the objects as per the role played.

This system created can be viewed using the drag and drop interface by running the command:

`ant standalone-xml` (As we are using xml persistor)

Also it can be seen using a HTML viewer by running the command:

`ant html standalone-xml` and then clicking on the following link:

`http://localhost:8080/logon.app`

6. ADVANTAGES AND DISADVANTAGES

Advantages

1. A faster development cycle

   Conventional applications require 4 separate layers namely presentation layer, controller layer, domain model layer and a persistence layer to be developed. But using Naked Objects all except behaviorally rich object layer is auto created from it. Thus, it requires development of fewer layers.

2. Greater Agility

   NOF supports flexibility in adding future changes in business requirements by limiting the number of developed layers that need to be synchronized. It improves agility due to higher quality domain object modeling. This is achieved by enforcing 1:1 correspondence between user presentation and domain model. Few requests for changes can also be accommodated in real time, hence taking rapid prototyping to a new level. Since the prototype itself documents the

object model, time spent in creating, editing and maintaining documentation is saved.

3. More empowering style of user interface

   High levels of modality lead to problems for the users. This is because they support 'verb-noun' style of interaction where the user selects the task before the data. Naked Objects automatically create OOUI that follow a 'noun-verb' style where users have select options for tasks on objects thus avoiding accidental misuse of data.

4. Easier requirements analysis

   Domain objects form a common language between the user and the developer thus helps in better and clearer understanding of the requirements. Due to faster development cycle, it is possible to prototype functional applications in real time using Naked Objects.

Disadvantage

NOF is suited for sovereign applications but not for transient ones.

7. CONCLUSION

Being behaviorally rich, Naked Objects Framework is a complete Object Oriented User Interface gaining popularity due to its visual appeal and agility that allows flexibility and supports various enhancements in the requirements phase. Both the developer and the customer can work on the system real-time, making it more interactive and the instant visual result keeps the customer satisfied. Since the interface is auto generated and does not require hundreds of lines of code to be written, it is gaining interest even among non-IT users as it does not require high level technical know-how.

In today's world where time is money and competition is high, the market requires open source tools that provide instant results and that appeal even to non technical users. Naked Objects Framework serves as such a perfect solution for Rapid Prototyping that has immense scope in the future.

8. ACKNOWLEDGMENTS

We would like to express our gratitude to Usha Mittal Institute of Technology and Patni Computer Systems Ltd. for encouraging us to take up this project activity as a part of the curriculum.

9. REFERENCES

[1] Richard Pawson and Robert Matthews, "Naked Objects", John Wiley & Sons Ltd., 2002.

[2] "Naked Objects", May 2009 [Online]. Available: http://www.nakedobjects.org/home/no_for_java_intro.shtml [Accessed: May 14, 2009].

[3] Siegfried Bolz, "Create CRUD-Mashups rapidly with Naked Objects and NetBeans", May 2008 [Online]. Available: http://blog.jdevelop.eu/2008/05/14/create-crud-mashups-rapidly-with-naked-objects-and-netbeans/ [Accessed: June 10, 2009].

[4] "Naked Objects 3.0: Integration with Hibernate", Sept 2006 [Online]. Available: http://www.theserverside.com/tt/articles/article.tss?l=NakedObjectsHibernate [Accessed: June 10, 2009].