Domain Modeling for MDA Using Activity Diagrams

Srinivas Nedunuri,
COMSYS Consultant,
Texas Dept. of Human Services,
Austin, Texas, 78751
srinivas.nedunuri@dhs.state.tx.us
Outline

- Introduction to software reuse at TDHS
- MDA
- Domains and Domain Engineering
- Specifying inter-object behavior
- Activity Diagrams
- Example: Application Security and its sub-domains
- Transforming PIMs to PSMs
- Issues regarding a full blown code generator
- Looking ahead
Software Reuse at TDHS

- A long term initiative to develop reusable assets that can be reused across a wide variety of applications across the State (and beyond).
- First area to be chosen: Application Security
The Model Driven Architecture (MDA) Initiative

“A complete MDA specification consists of a definitive platform-independent base UML™ model (PIM), plus one or more platform-specific models (PSM) and interface definition sets, each describing how the base model is implemented on a different middleware platform.”
Basic Idea

\[ \{\text{PIM}\} \times \{\text{PDM}\} \implies \text{PSM} \]

Example:

Loan Processing + Finance + Security + ... 
\times 
J2EE App Server + EJ Bs + Oracle DB + LDAP + Web Browser 
\implies 
Loan Processing App

The PIMs (and PDMs) are *domain models*
Intent of PIM and PSM

- The PIM defines a complete model (the interface model) against which client domains will program.
- The PSM embodies that interface model in its implementation but it may not be easily visible (c.f. source -> assembly)
- This is true separation of interface from implementation
Outline

✔ Introduction to software reuse at TDHS
✔ MDA
- Domains and Domain Engineering
- Specifying inter-object behavior
- Activity Diagrams
- Example: Application Security and its sub-domains
- Transforming PIMs to PSMs
- Issues regarding a full blown code generator
- Looking ahead
What is a Domain?

Domain = An area of knowledge

- a set of concepts and terminology understood by practitioners in that area

+ knowledge of how to build software in that area
Domain Definition and Scope

A body of knowledge is only a domain to the extent that it is useful to a community of practitioners in that area. It is constantly being refined and added to.

A domain must be maintained (and therefore maintainable!) over time.
Example Domains in an Implementation Hierarchy

- **Application Domains**
  - Accounting
  - HR
  - Shopping Cart

- **Technology Domains**
  - Web Architecture
  - GUIs
  - Events
  - Security
  - Scheduling

- **Implementation Domains**
  - State Models
  - Collections
  - Hash Tables
  - Trees
  - Graphs

- **Language Domains**
  - Java
  - Haskell
  - Prolog
  - C
So how are domains used?

- Every application can be considered as the composition of one or more domains
- Domains can be reused across a variety of applications
- We will need a library of domains
- The various OMG Domain Task Forces are busy constructing domain models for a number of pervasive domains
Stages of Domain Engineering

- Domain Analysis
  - Identify and scope the domain(s) and their dependencies (Domain Scoping)
  - Identify features (Feature Analysis)

- Domain Modeling
  - Construct static and behavior models of each domain (= PIMs)

- Domain Implementation
  - Map them to the platforms to get a PSM
Feature Oriented Domain Analysis (FODA)

Describe the features of each domain, how the features relate to each other, and what the purpose and binding time† is of each feature.

NOTE: features are externally visible aspects that are of interest to a user of the domain.

† When the choice of feature is bound in
Example feature analysis

- Concept
- Mandatory feature
- Exclusive alternatives
- Optional feature
- Inclusive alternatives

From *Generative Programming: Methods, Tools, and Applications* by Czarnecki and Eisenecker, © Addison Wesley, 2000
What next?

At this point, a class diagram may be drawn, but then most modeling tails off, either resorting to informal description or descending straight down to code. A few Interaction diagrams might be drawn or an API might be published. But this is not precise enough for MDA.

Where has the behavior been defined?
There is a better way...

Instead of drawing Interaction Diagrams, we believe it is better to “open” up the behavior model, the same way that the static information is available in an open model (the Class Diagram).
Outline

✔ Introduction to software reuse at TDHS
✔ MDA
✔ Domains and Domain Engineering
  ▪ Specifying inter-object behavior
  ▪ Activity Diagrams
  ▪ Example: Application Security and its sub-domains
  ▪ Transforming PIMs to PSMs
  ▪ Issues regarding a full blown code generator
  ▪ Looking ahead
Modeling using Implicit Invocation

Implicit invocation (or event based) architectures are described in Shaw and Garlan’s book “Software Architecture”. Their benefits are:

- Inherently concurrent
- Support loosely coupled architecture
- Support an “open” model
State Models

- State models are excellent for defining the behavior of instances of particular classes, and are used extensively in the Shlaer Mellor method and its descendents.

- However, sometimes the interesting behavior is in the *interactions* between the objects, and not in the objects themselves.

  ➤ Introducing Activity Diagrams…
Example Activity Diagram

Figure 3-91  Actions and Object Flow
Activity Diagrams: Benefits

- They represent implicit invocation architectures nicely
- Graphical, and easy to grasp
- They have a clean implementation in event driven standards (such as Java Beans or Message Driven EJBs)
- Concurrency and Asynchronicity are inherent
Activity Diagrams

- Activity diagrams have been used mainly for informal modeling of the sort illustrated previously. They have also been used to illustrate the flow of use cases [UML Distilled by Fowler & Scott].

- It is possible to use Activity diagrams in a more formal manner to model the inter object behavior of domains.
Extensions to UML Activity Diagrams notation

- Extensions made to visual representation (not the semantics). This is consistent with the UML spec which distinguishes activity diagrams from activity graphs.

- The extensions will be pointed out during the talk.
Outline

- Introduction to software reuse at TDHS
- MDA
- Domains and Domain Engineering
- Specifying inter-object behavior
- Activity Diagrams
  - Example: Application Security and its sub-domains
  - Transforming PIMs to PSMs
  - Issues regarding a full blown code generator
- Looking ahead
Security Sub Domains†

- Authentication
- Login
- Security Attributes
- Access Control (includes Authorization)
- Auditing
- Metasecurity
- Confidentiality & Integrity

[†] Based on requirements drawn from the Common Criteria (www.commoncriterion.org/faq/faq.html)
Scoping the Security domain

Core Security Boundary

Service Domains
The Authentication domain
The Authentication domain

Behavior Model

AuthenticationServer.authenticate(Id uid, Password pwEntry)

equals(pwEntry)

Authenticated(uid)

NotAuthenticated(uid)

securityData.Directory.lookupPassword(uid)

pW

PasswordExpired

AccountDisabled

UserIdNotFound

Sender Address Verification is implemented by the Transport Level Protocol domain (e.g. using IPSec)

denotes pure action (side effect free)

object flow

call state

the target of the call

retransmitted event

exception
The Authentication domain

Class Model

Authentication Server

authenticate(id : Id, password : Password) : void
The Security Data domain
The Security Data domain

Class Model

- `Subject`
  - `id : Id`
  - `currentStatus : Status`

- `Security Attribute`
  - `name : String`
  - `description : String`

- `Role`
  - `subject : Subject`
  - `subroles : Role`

- `Group`
  - `members : Group`
  - `subgroups : Group`

- `CurrentApp`
  - `principals : Principal`

- `Principal`
  - `name : String`
  - `description : String`

- `Password`
  - `length : int`
  - `age : Time = 0`
  - `equals(anotherPw : Password) : boolean`

- `Policy`
  - `passwordHistory : Password`

- `Directory`
  - `lookupPasswordFor(userId : Id) : Password`
  - `setPasswordFor(userId : Id, newPw : Password) : void`
  - `lookupSubject(userId : Id) : Subject`

- `Password`
  - `equals(anotherPw : Password) : boolean`
  - `length : int`
  - `age : Time = 0`

- `Policy`
  - `current : Password`

- `Observes`
The Security Data domain

**Behavior Model**

```java
void DataManager.changePassword(Id uid, Password currentPw, Password newPw)

Policy.
checkCompliance
(currentPw, newPw)

Y

Directory.
SetPasswordFor
(uid, newPw)

Directory.
lookupSubject(uid).
passwordHistory.
add(newPw)

PasswordChanged

[N]

PasswordNotChanged

Password Directory.lookupPasswordFor(Id uid)

getPasswordFor
(uid)

currentPw

Y

currentPw.age >
Password.EXPIRATION_PERIOD

PasswordExpired(currentPw)

UserIdNotFound

AccountDisabled
```
The Login domain

Feature Model
The Login domain

Behavior Model
The Login domain

Class Model
The Authorization domain

Feature Model

Constraints: If "Run As" is not chosen, then "Delegation" is a mandatory feature
The Authorization domain
(access control list based)

Type of Access Control: No specific class structure. Implemented by interfacing to application logic (programmatic) or by generating the interface to application logic (declarative)

```
Map<Object x Set<securityData.Role> x String, Boolean>
```

Class Model
The Authorization domain (access control list based)

currentPrincipal: subjectInfo.Principal

roles: Set<subjectInfo.Role>

restrictedResources.contains (e.resource)

currentPrincipal holds, at runtime, the principal making the current access to the resource

e:BeforeMethod

ACL.lookup(e.resource, e.methodName, roles)

(Map<Object x String x Set<subjectInfo.Role> Boolean>)

access?Y

AccessGranted (e.methodName, resource)

AccessDenied (e.methodName, resource)

Behavior Model
The Authorization domain (access rights based)

Type of Access Control: No specific class structure. Implemented by interfacing to application logic (programmatic) or by generating the interface to application logic (declarative)

Class Model
The Authorization domain (access rights based)

- **currentPrincipal**: subjectInfo.Principal
- **roles**: Set<subjectInfo.Role>

Determine **GrantedRights** (AccessPolicy)

- **superset?**
- **access?**

*If access is granted*

- **AccessGranted** (e.methodName, resource)

*If access is denied*

- **AccessDenied** (e.methodName, resource)

**Behavior Model**

*If access is determined*

- **appResource**, reqdRights{methodName}: Set<subjectInfo.Role>

*If access is determined and granted rights exist*

- **grantedRights**: Set<Right>

*If access is determined and rights are not granted*

*If access is determined and rights are not granted*

- **isRestrictedMethod(e.methodName)**

*If currentPrincipal holds, at runtime, the principal making the current access to the resource*
The Auditing domain

Feature Model
1. currentPrincipal holds, at runtime, the principal making the current access to the resource
2. AfterAuditMethod extends AfterMethod with extra attrs
The Auditing domain

AuditTrail
- #MAX_LENGTH : int
- Sort() : void
- Search(criteria : LogicExpression) : List<AuditRecord>
- filter() : void

AuditRecord
- +signed : Boolean

EventDetail
- +name : String
- +date : Date
- +time : Time
- +identity : subjectInfo.Principal
- +outcome : Outcome

Outcome

Filter
- +criteria : LogicExpression

AuditEventType

AuditableEventType

Class Model
The Metasecurity domain

Feature Model
The Confidentiality & Integrity domain

Feature Model
The Metasecurity and C&I domains

Neither of these domains have interesting class or behavior models. Rather they are implemented by procedural steps or by lower level domains (e.g. using IPSEC for the Transport Level Protocol domain to prevent replay attacks).
This is visual modeling, NOT visual programming!

- Notice that the algorithmic, platform specific parts of the model (the method bodies) are not described graphically.
- They are filled in separately, either directly with code (Java), or in an Action Semantics† language

† an extension to UML 1.4
Outline

- Introduction to software reuse at TDHS
- MDA
- Domains and Domain Engineering
- Specifying inter-object behavior
- Activity Diagrams
- Example: Application Security and its sub-domains
  - Transforming PIMs to PSMs
  - Issues regarding a full blown code generator
  - Looking ahead
Approach

- PIM -> PSM is done in two stages:
  - PIM -> Base
  - Base -> PSM

- PIM -> Base: Generate the APIs + base implementation as a framework. Currently looking into a code generator for this

- Base -> PSM: Code the PS parts in the methods
Example Base API

The following slides show the API and implementations that could be automatically generated from the Authentication domain models

```java
package tdhs.security.authentication;

import java.util.EventObject;
import tdhs.security.securityData.*;
import java.util.Vector;

/**
 * Provides methods and events to support authentication of a potential user
 * identified by an <code>id</code>.
 * Depends on password based authentication.
 * An interface to be implemented by a singleton class.
 * Creation date: (7/25/2002 6:11:02 PM)
 * @author: Srinivas Nedunuri
 */
public interface AuthenticationServer
{
    public void authenticate(Id id, Password pwEntry);
    public Vector getAuthenticationEventListeners();
    public java.util.Vector getPasswordEventListeners();
}
```
Example Base Implementation

```java
package tdhs.security.authentication;

import java.util.EventObject;
import java.util.Vector;
import tdhs.events.*;
import tdhs.exceptions.*;
import tdhs.security.securityData.*;

/**
 * Implements the AuthenticationServer interface and Authentication domain model
 * @creationDate (9/20/2002 12:23:46 PM)
 * @author S Nedunuri
 */
class AuthenticationServerImpl implements AuthenticationServer
{
    Vector authenticationEventListeners = new Vector();
    Vector passwordEventListeners = new Vector();

    public AuthenticationServerImpl()
    {
        super();
    }

    // <authenticate, getAuthenticationEventListeners, and
    // getPasswordEventListeners on next slide>

    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {
        throw new NotImplementedException();
    }
}
```
public void authenticate(Id id, Password pwEntry) throws AccountDisabledException, UserIdNotFoundException
{
    Password pW;
    Directory directory = tdhs.security.securityData.Factory.instance.getDirectoryInstance();

    /* this next part would not even be generated by a code generator since it would directly descend
    the event subscription nesting in the model, and carry this out itself (directly hook this class's
    PasswordEventListener to the Directory object's list of listeners, bypassing this class entirely)*/
    directory.getPasswordEventListeners().add(new tdhs.security.securityData.PasswordEventAdaptor());
    for(int i=0; i<passwordEventListeners.size(); i++)
        directory.getPasswordEventListeners().add(passwordEventListeners.get(i));

    try
    {
        pW = directory.lookupPasswordFor(id);
    }
    catch(AccountDisabledException e)
    {
        throw e;
    }
    catch(UserIdNotFoundException e)
    {
        throw e;
    }

    AuthenticationEvent e = new AuthenticationEvent(this);
    e.id = id;

    for(int i=0; i<authenticationEventListeners.size(); i++)
    {
        if(pwEntry.equals(pW))
            ((AuthenticationEventListener)authenticationEventListeners.get(i)).Authenticated(e);
        else
            ((AuthenticationEventListener)authenticationEventListeners.get(i)).NotAuthenticated(e);
    }
Base Implementation contd:

get<Named>EventListeners

/**
* @return the list of current <code>AuthenticationEventListener</code>s.
* To subscribe to <code>AuthenticationEvent</code>s, add your implementation of <code>AuthenticationEventListener</code> to this list. NB: Listeners that are added to this list in the midst of a broadcast are not guaranteed to receive the event. Listeners added more than once will receive the event more than once.
*/
public java.util.Vector getAuthenticationEventListeners() {
    return authenticationEventListeners;
}

/**
* @return the list of current <code>PasswordEventListener</code>s. NB These are just passthru events sent from the domain below
* To subscribe to <code>PasswordEvent</code>s, add your implementation of <code>PasswordEventListener</code> to this list. NB: Listeners that are added to this list in the midst of a broadcast are not guaranteed to receive the event. Listeners added more than once will receive the event more than once.
*/
public java.util.Vector getPasswordEventListeners() {
    return passwordEventListeners;
}

The code for the more complex domain models such as Login and Authentication will be correspondingly longer and more detailed!
The PSM

The platform specific implementation would provide implementation of the method `directory.lookupPasswordFor(Id)` which in turn would call `directory.getPasswordFor(Id)`. This method is implemented by making calls Java LDAP calls on a directory service such as NDS.
Outline

- Introduction to software reuse at TDHS
- MDA
- Domains and Domain Engineering
- Specifying inter-object behavior
- Activity Diagrams
- Example: Application Security and its sub-domains
- Transforming PIMs to PSMs
  - Issues regarding a full blown code generator
  - Looking ahead
Difficulties in the way of a full blown code generator

- The huge number of combinations of implementation technologies in use (e.g. choice of web server x choice of database x choice of directory server x choice of messaging x choice of component model) makes it difficult to cover them all.
Difficulties (contd.)

- Non-local effects of replacing one technology piece by another. E.g. replacing say the web server has implications for whether or not you use EJBs, which may affect how you do security. Also, different vendors implement different pieces of the puzzle (some databases offer their own security solution)
At the Dept of Human Services, we are in the process of standardizing on a target system architecture:

- NDS (via LDAP) for directory services
- J2EE compliant web server (IBM Websphere 4.x) in Struts framework
- Oracle database for backend
- (for larger apps) Novell Secure Login for SSO
- others?
Even so...

- Cleverness will be needed in implementing (e.g. J2EE Design Patterns)
- Support for alternatives (CORBA access to EJBs, open source, or Web Services)
- Some apps will be “lighter” than others
- Technology changes (e.g. smart cards, PDA access, vendor upgrades)
- Incompleteness in “specs”. E.g. EJB Security: leeway to the container provider
Summary of the technical challenges in MDA

- How to transform the domain models to efficient implementations on a variety of target platform configurations, when the structure of those configurations can be quite different from the structure of the original domain models?
- Composition of domain models (c.f. aspect oriented modeling?)
For effective reuse...(the ultimate purpose of MDA)

- Need to change processes. Existing processes are based on the old waterfall style for developing one off applications
- Team organization and structure
- Incentives for reuse
- Asset certification & revision process
- Library & admin