



Transferring Data Between Remote Clients and Analyzed Domains Running on the Server

Carolyn Duby

Version 1.2
April 22, 2009

Pathfinder Solutions LLC
33 Commercial Street, Suite 2
Foxboro, MA 02035 USA
www.PathfinderMDA.com
888-662-7284

Table Of Contents

- 1. Introduction 3
- 2. Transfer Object Assembler Pattern 3
- 3. Mapping Analyzed Classes to Transfer Objects..... 4
 - Define Bundles 4
 - Access Transfer Objects in Platform Independent Models 7
- 4. Feature Details..... 8
- 5. Markings 9
 - Class 9
 - Participant 9
 - User Defined Type..... 9
- 6. References 9

1. Introduction

This paper describes how data transferred between client and server can be optimized. We have two main concerns:

- Minimizing network traffic between client and server.
- Minimizing data coupling between the client and server.
- Centralizing business logic on the server.

2. Transfer Object Assembler Pattern

The Core J2EE Patterns book describes the Transfer Object Assembler pattern (pg. 433 – 443). The pattern uses a transfer object assembler to compose a set of transfer objects and returns the transfer objects to the client. A transfer object "carries multiple data elements across a tier" (pg. 415).

When the client needs data from the server, it makes a request. The server assembles the transfer objects using data from the database and returns the transfer objects back to the client. Alternatively the client could get the transfer objects through a subscription service. The client updates the GUI with the new data.

The transfer objects are basic Plain Old Java Objects (POJOs) containing only data and implementing the `java.io.Serializable` interface. The transfer object assembler is the stateless session bean generated from the services provided by a domain. Markings in the model and a CSV configuration file describe how to map data stored in class instances to transfer objects.

The Transfer Object Assembler has the following positive consequences:

- Removes business logic from the client tier.
The logic of assembling the transfer objects is centralized on the server.
- Reduces coupling between client and server
The data stored by the server can change but as long as the transfer objects stay the same, the client does not need to change.
- Improves network performance
Reduces the number of roundtrips required for the client to get the data it needs from the server.
- Reduces client resources
The client makes a single call to access the data rather than multiple calls.

The Transfer Object Assembler pattern has the following negative consequences:

- Clients receive a snapshot of server data so the data may become stale.

3. Mapping Analyzed Classes to Transfer Objects

Define Bundles

Bundles define which classes to include in the transfer objects. Bundles are a set of classes interconnected by associations. Each bundle has a unique name.

A bundle has a root class. The root class may have one or more branches defined by associations. The branches connect the root to members. Members may connect via branch associations to other members forming a tree structure.

A class or association may participate in more than one bundle. Within the same bundle, a class may be either a root or a member but not both.

For example, Figure 1: Entity Bundle shows a bundle with the Entity class at the root in blue. The members of the bundle are shown in white – Observation and PredictedZone. The branches of the bundle are shown in black – A2 and A5. Figure 2: Entity Bundle Markings shows the markings used to specify the bundle.

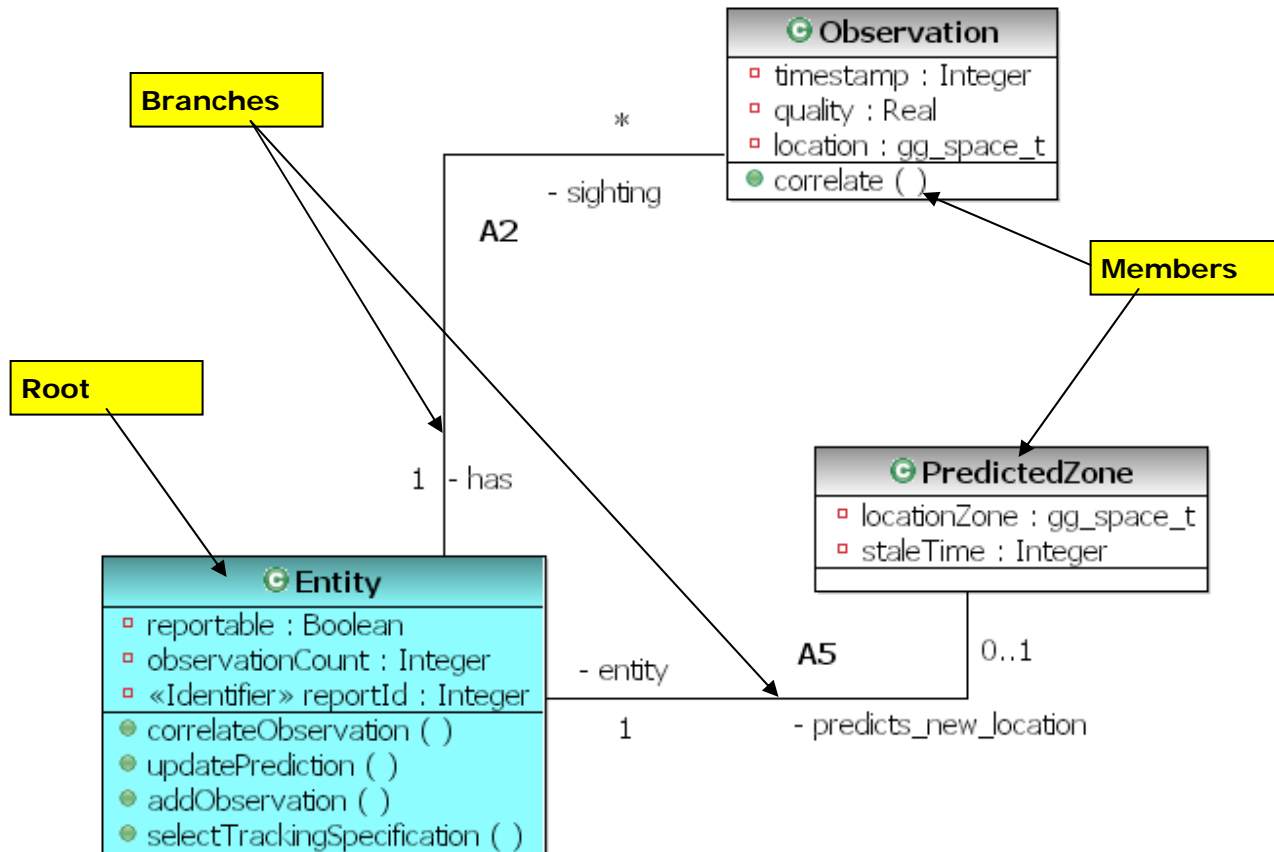


Figure 1: Entity Bundle

```
# bundle root
Object,Sentry.EntityTracking.Entity,BundleRoot,EntityBundle
# set EntityBundle to a transfer object
Object,Sentry.EntityTracking.Entity,BundleType,EntityBundle=Transfer
# bundle members
Object,Sentry.EntityTracking.Observation,BundleMember,EntityBundle
Object,Sentry.EntityTracking.PredictedZone,BundleMember,EntityBundle
# bundle branches
Participant,Sentry.EntityTracking.A2.Entity.has,BundleBranch,EntityBundle
Participant,Sentry.EntityTracking.A5.Entity.entity,BundleBranch,EntityBundle
```

Figure 2: Entity Bundle Markings

Map Bundles to Transfer Objects

After defining the bundles, define how the classes, attributes, and associations of the bundle map to transfer objects. A Comma Separated Value (CSV) file defines the mapping between the bundles and transfer objects. The transformation map generates an initial spreadsheet in a subdirectory of the generated code. Copy the initial spreadsheet to the deployment working directory and modify it using a text editor or Excel.

Each bundle must have a CSV file named <bundle name>TransferObjects.csv in the deployment working directory where the properties.txt is located. The CSV file has the following format for each class:

TransferObject,<class name>,<transfer object name>,<inst filter op>

Attr, <attribute name>,<transfer object field name>

Assoc, <association id>,<transfer object field name>

CSV Field Name	Meaning
<class name>	The unqualified name of a class which is either a root or member of the bundle.
<transfer object name>	The fully qualified name of the transfer object class that will represent this class.
<inst filter op>	Optional. The name of an operation of <class name>. The operation must be an instance based operation that returns a Boolean and takes no parameters. If the operation returns TRUE, the instance will be included in the transfer object. If the operation returns FALSE, the instance will be omitted from the transfer object.
<attribute name>	The name of the attribute of the class.
<association id>	The identifier of the association of the class, i.e. A<N>. If the association is reflexive (both participants are the same class) use A<N>.<role_name>.
<transfer object field name>	The name of the field in the transfer object that the attribute maps to. If the attribute is to be excluded from the transfer object, set this column to <exclude>.

Access Transfer Objects in Platform Independent Models

To use a transfer object in a model, define a UML Primitive Type to hold the bundle in the Domain Types folder of the domain package containing the classes comprising the bundle. On the Advanced Properties tab set PathMATE > Base Type to 3 – Handle. Set PathMATE > External to True.

In the properties.txt file, set the BundleName marking of the user defined type to the name of the Bundle carried by the type. The

templates set automatically the ExternalType property to the Transfer Object of the root class of the bundle.

On a domain service, set the return type or an output parameter type to the user defined type for the bundle. Set the return value or output parameter to an instance of the bundle root. The templates will generate the code to convert the bundle to the transfer object.

For example, suppose we defined a primitive type called `et_entity_t` and we have a domain service called `GetFirstEntity` that returns a `et_entity_t`, the action language would look something like this:

```
// get an instance of the root class of the bundle using a
// FIND or other navigation.
Ref<Entity>    entity = FIND CLASS Entity;

// convert the entity to its transfer objects
// and return it
RETURN entity;
```

To define a set of bundles, create a group of user defined types. Set the return type or output parameter to the group type. Add elements to the group and then return it.

In our example create the user defined type `Group<et_entity_t>`. Suppose we have a domain service called `GetAllEntities` that returns a `Group<et_entity_t>`. The action language would look something like this:

```
// create a list of transfer objects
Group<et_entity_t>    all_entities;
Ref<Entity>           entity;
FOREACH entity = CLASS Entity
{
    // convert entity to its transfer objects
    // add the transfer object to the list.
    all_entities.addBack(entity);
}

RETURN all_entities;
```

4. Feature Details

This feature builds upon templates already in PathMATE supporting bundles and CSV files. The templates will generate the following:

- Transfer object Java class definitions
- Java methods to convert class instances into transfer objects
- Invocations of conversion methods when a bundle root class instance is assigned to its transfer object user defined type

- Initial spreadsheets to define the mappings between bundle classes and transfer objects

5. Markings

Class

Marking Name	Default Value	Effect
BundleRoot	<blank>	A non-blank value indicates that this class is the root of the named bundle. If the class is the root of multiple bundles, use a semicolon separated list of bundle names.
BundleMember	<blank>	A non-blank value indicates that this class is a member of the named bundle. If the class is a member of multiple bundles, use a semicolon separated list of bundle names.
BundleType	<blank>	The type of bundle rooted in this class. Consists of a semicolon separated list of <bundle_name>=<bundle_type> where bundle_type is either Tasking or Transfer. Transfer is the default.

Participant

Marking Name	Default Value	Effect
BundleBranch	<blank>	A non-blank value indicates that this participant is a branch of the named bundle. If the participant is a branch of multiple bundles, use a semicolon separated list of bundle names.

User Defined Type

Marking Name	Default Value	Effect
BundleName	<blank>	A non-blank value indicates that this user defined type is a transfer object for the specified bundle name.

6. References

Alur, Deepak, John Crupi, and Dan Malks. *Core J2EE Patterns Best Practices and Design Strategies (Second Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, 2003.

Burke, Bill and Richard Monson-Haefel. *Enterprise JavaBeans 3.0
(Fifth Edition)*. O'Reilly, Cambridge, 2006.