# C++ Architecture Users Guide

version 0.1
9/13/02

**1. INTRODUCTION** .......... **3**

**2. OVERVIEW** .......... **4**

**3. QUICK START** .......... **4**

**4. TARGET DOCUMENT SET** .......... **4**

4.1 DIRECTORY HIERARCHY .......... 4
4.2 FILES .......... 4
4.3 SCHEMAS .......... 4

**5. MECHANISMS** .......... **4**

**6. STRUCTURAL DESIGN** .......... **5**

6.1 HARDWARE .......... 5
6.2 SOFTWARE .......... 5

**7. PROPERTIES** .......... **6**

**8. BUILD SETTINGS** .......... **9**

**9. MAPPINGS** .......... **10**

9.1 DOMAIN CHART CONSTRUCTS .......... 10
*9.1.1 Bridge .......... 10*
*9.1.2 Constant .......... 10*
*9.1.3 DataType .......... 10*
*9.1.4 UserDefinedType .......... 10*
*9.1.5 Domain .......... 11*
*9.1.6 System .......... 12*
9.2 INFORMATION MODELING CONSTRUCTS .......... 12
*9.2.1 Attribute .......... 12*
*9.2.2 UML Class .......... 12*
*9.2.3 Relationship .......... 13*
9.3 STATE AND SERVICE MODELING CONSTRUCTS .......... 14
*9.3.1 Action .......... 14*
*9.3.2 Event .......... 15*
*9.3.3 Generate .......... 15*
*9.3.4 State Transition Table .......... 15*
*9.3.5 Output Parameters .......... 15*
9.4 ACTION LANGUAGE .......... 16
*9.4.1 AttributeSelection .......... 16*
*9.4.2 Create .......... 16*
*9.4.3 CreateServiceHandle .......... 16*
*9.4.4 Delete .......... 16*
*9.4.5 Expression .......... 16*
*9.4.6 Instance Lookups - Find/Foreach .......... 16*
*9.4.7 InvokeServiceHandle .......... 17*
*9.4.8 Link .......... 17*
*9.4.9 Navigation .......... 17*
*9.4.10 ServiceInvocation .......... 17*
*9.4.11 SubSuperNavigation .......... 18*
*9.4.12 Unlink .......... 18*
9.5 MBSE SEMANTICS .......... 18

*9.5.1 Timers* ... *18*
*9.5.2 Instance Lists* ... *18*
*9.5.3 Service Handles* ... *18*
*9.5.4 Task* ... *19*

**10. IMPLEMENTATION SPECIFIC ISSUES ... 19**

*10.1.1 Exceptions* ... *19*
10.2 EJB SPECIFIC ISSUES ... 19
*10.2.1 Loopbacks* ... *19*
*10.2.2 Exceptions* ... *19*
*10.2.3 Transactions* ... *19*
10.3 REALIZED INTERFACES ... 20

**11. PORTING ... 21**

**12. MODELING CONVENTIONS AND RESTRICTIONS ... 22**

**13. TERMS AND DEFINITIONS ... 22**

**14. REFERENCED DOCUMENTS ... 22**

| **Version** | **Date** | **Person** | **Comments** |
|---|---|---|---|
| **0.1** | **9/13/02** | **Greg Eakman** | **Created** |

# 1. INTRODUCTION

Model Based Software Engineering (MBSE), as presented by Pathfinder Solutions, is a rigorous, complete, unambiguous, high level executable model of a particular problem domain. As such, this model can be translated into other executable implementations in the form of source code.

Model data is stored in an MBSE meta-model database. A set of translation rules in the form of ASCII templates, read and interpreted by Springboard, produces the source code. Springboard is a translation engine that extracts the semantic information contained in your UML analysis models and presents it in textual form via a flexible and simple template notation. See the Springboard Users Guide for more information on what data is accessible through the templates and the format and structure of templates.

This document describes the mapping of analysis into a XXX design using Springboard and translation templates. The reader is assumed to have a working knowledge of Model Based Software Engineering (MBSE). The reader is also assumed to be familiar with the Unified Modeling Language (UML™) and the implementation language.

It is also assumed the reader is familiar with the terminology, concepts and conventions presented in:
"Model Based Software Engineering – an Overview of Rigorous and Effective Software Development Using UML™" 1998, Pathfinder Solutions, Inc.

The Overview Section describes the overall design philosophy of the architecture and its applications. A Quick Start section provides first time users and evaluators with the set of steps to generate, build, and execute a model for the first time. It is assumed that the reader is familiar with the editor environment and has read the editor specific modeling guides, or is working with a Pathfinder provided sample model.

The Target Document set describes the set of artifacts produced by the translation rules to fulfill the architectural requirements. The Properties section describes the properties that can be applied to UML model elements, their possible values, meaning, and effect on the generated code. The Mechanisms section describes the hand-written code that supports the execution of the models. The Structural Design section describes some of the possible structural designs supported off-the-shelf, such as multi-threaded, distributed, etc. The Build Settings section describes the compile-time build switches and their effect on the generated application.

The Mappings section is organized by the major constructs within the MBSE meta-model. A brief description of each analysis construct is provided, followed by the mapping rules used to generate the implementation. When appropriate, the translation rule pattern is shown in a UML diagram. Where choices in the implementation can be made, properties which guide the templates on how to translate the construct are referenced. Optimizations inferred by the translation templates based on the models themselves are described as well. The templates which contain the translations rules are listed for each UML element.

Implementation specific issues are covered in their own section. The Porting section describes issues that should be addressed to port the code to other operating systems or environments. The section provides a list of platforms to which the architecture has already been ported. Finally, the Modeling Conventions and Restrictions section describes any limitations the architecture places on the models themselves.

## 2. Overview

## 3. Quick Start

This Quick Start section provides first time users and evaluators with the set of steps to generate, build, and execute a model for the first time. It is assumed that the reader is familiar with the editor environment and has read the editor specific modeling guides, or is working with a Pathfinder provided sample model.

## 4. Target Document Set

Considering the task of translation at the highest level, the goal is to establish a set of instances of target documents with a set of analysis data according to specified templates, or templates. The templates specify the specific analysis information required, how it's arranged, and in what context.

The above paragraph helps us identify the major tasks/components of a translation:

- Develop a set of templates to define our target document set
- Employ Springboard to validate the correctness of the new templates
- Apply the proven templates with Springboard to produce the Target Document Set

The target document set consists of a set of generated files, organized in a directory structure. The target document set may include:

- generated source code
- realized source code
- deployment descriptors
- static data initialization
- schemas
- build files
- IDE configuration files.

### *4.1 Directory Hierarchy*

### *4.2 Files*

### *4.3 Schemas*

## 5. Mechanisms

Pathfinder Solutions provides a set of mechanisms, in the form of source code. These mechanisms provide the classes required to execute the MBSE semantics within the target architecture.

# 6. Structural Design

This document mainly addresses the Mechanical Design aspect of system generation, translation rules and supporting mechanisms. The a general structural design consisting of the hardware layout, the software technologies, and component allocation in the distributed environment is presented here as context for the translation rules.

## *6.1 Hardware*

## *6.2 Software*

## *6.3 Communication Threads*

## *6.4 Interfacing with Realized Code*

## 7. Properties

| Name | Scope | Values | Description |
|---|---|---|---|
| ProcessorIds | System | proc1;proc2 | A “;” separated list of processor IDs used in the system deployment |
| SupportFiles | System | directory name | indicates where the PAL files and TYP files are found, and where to put WMF file during document generation |
| Prefix | Domain | String | standard MBSE naming property used for shorthand reference to the domain. Default value is the full domain name. |
| ProcessorId | Domain | From the list of system ProcessorIds | indicates what processor this domain is allocated |
| Realized | Domain | T or F | If T, only the domain services interface will be generated |
| SpotlightEnabled | Domain | T or F | indicates instrumentation is to be generated |
| SupportFiles | Domain | directory name | allows a domain's support files to be distinct from the system area |
| Prefix | Class | String | standard MBSE naming property used for shorthand reference to the class. Default value is the full class name. |
| MaxIndex | Class | Integer | if this is specified this indicates an array structure is to be used for this class' instances, specifying the array upper bound. |
| Include | Class | Text | Indicates any special include files required by this class |
| ProcessID | Class | From the list of system ProcessorIds | Used to aid in multi-process deployment (C++ only) |
| SortOrder | Class | | Indicates how the class instance tracking structure is to be sorted |
| SortKey | Class | | Indicates the key attribute for above if order is ascending or descending |
| DefaultValue | Attribute | Depends on the type of the attribute | default value assigned to the attribute when a new instance of the class is created. |
| ExplicitDelete | Attribute | T or F | indicates this attribute is a pointer to something that must be deallocated |
| Identifier | Attribute | T or F | used by Spotlight to identify this instance to the user |
| Derived | Service | T or F | (only for domain services): indicates that a template will expand this service body, like a macro inline expansion |
| Ignored | State | Text | list of events that are ignored (dropped) in this state |
| Deferred | State | Text | list of events that are deferred (requeued) in this state |
| IncludeFile | VarDeclaration | filename | Property defined in Action Language, indicates any special include files required by this declaration |
| Serializable | DataType | T or F | (C++ only) indicated this inherits from PfdSerializable for communication purposes |
| | | | |
| | | | |

| | | | |
|---|---|---|---|
| controlPoint | AL Statement | String | Application specific property to notify spotlight of a named breakpoint. |
| CreateBitfield | Class | T or F | Internally managed property, derived from the EnableBitfields domain property. Is only set to true when Spotlight instrumentation is off. |
| StaticPopulation | Class | T or F | T indicates the class extent is static and the table of its instances will be defined. |
| Used | GroupTyp e | T or F | Internally managed property to capture if a group type is used or not. |
| IE_Enabled, SpotlightEnabled | Domain | T or F | Enables Spotlight instrumentation for the domain. |
| EnableBitfields | Domain | T or F | Compresses boolean and enumeration attributes into a single bitfield and creates set/get methods for them. Only applied when instrumentation is off. |
| Multithreaded | Domain | T or F | Used to determine if thread safe lists should be used in class extents and associations. |
| SourceExtension | Class | Text | Inlines the text into the generated class .cpp file immediately after the #includes and before the class methods. Used to extend behavior of the class. Preferred setting is to use "#include classExtension.hpp" |
| HeaderInclude | Class | Text | Add text at the end of list of generated #includes. |
| LargePopulationClass | Class | Text | Defines the container class for large static instance populations. |
| ClassExtension | Class | Text | Inlines Text into the class .hpp file to extend behavior of class in application specific ways. Used with SourceExtension. Text is inserted ay bottom of class definition. |
| CleanUp | Class | Text | Application specific clean up code for the class. Text is added to the class destructor. |
| Include | Class | Text | Adds text at the end of the generated #includes |
| OptimizeUniNav | Assoc | T or F | Determines if an association end (Participant) can be optimized by removing the linking pointer/list. This is discussed more in the template rel_opt_lu.arc. |
| ThreadId | Domain Svc | CONTROL or SUPPORT | Early support for multi-threading. Phased out with general multi-thread support. |
| ReferencedType | Data Type | Text | Used to support external complex types that must be serialized across a net connection. |
| ReferenceCounted | Data Type | T or F | T if instances of the complex data type are reference counted. |
| ConstantConstants | System | T or F | Used to keep generated constants for service and parameter numbers the same across gencode operations. Uses an external file to store the constants, and needs an external utility executable, pfdMerge.exe, to manage that file. |
| RealizedPath | System | Text | Used for generating Visual C++ projects and workspace. A ";" separated list of directories containing realized code to include in the project. |
| SpotlightNetConnection | System | T or F | Used for generating Visual C++ projects and workspace. T uses a socket connection to Spotlight, F uses legacy shared file communications. |
| Defines | System | Text | Used for generating Visual C++ projects and |

| | | | |
|---|---|---|---|
| | | | workspace. A ";" separated list of symbols to define (compile time switches). |
| SysUmlPath | System | Text | Used for generating Visual C++ projects and workspace.  Path to the MBSE startup files. |
| GeneratedPath | System | Text | Used for generating Visual C++ projects and workspace.  Single path to the root of the generated code. |

## 8. Build Settings

This section describes the compile-time build switches and their effect on the generated application.

# 9. Mappings

This section is broken down by the into sections based on the following MBSE constructs, Domain Chart, Information Modeling, State and Service Modeling, and Action Language. While this breakdown overlaps somewhat, it is easiest to consider these as sections.

This section covers the large grained mappings for MBSE/UML to the target implementation. See the Springboard Users Guide for the full list of analysis elements accessible in the templates.

In cases where there are alternative translation rules that can apply, properties can be assigned to MBSE analysis constructs to guide the templates in the translation. Properties are name-value pairs associated with analysis elements, where the names are project-specific string constants. Properties can be specified in the model editor, or at translation time in the templates, using the SetProperty command in the Springboard Template Language. In each section, if there are possible alternative implementations, these are discussed, as well as the trade-offs, and property names are defined to guide the translation.

## *9.1 Domain Chart Constructs*

### 9.1.1 Bridge

supertype: none; subtypes: none; A requirement flow line connecting two Domains on the domain chart.

A bridge defines the requirements level relationship between domains, as well as the service invocations that are made from one to the other. There is no implementation reflection of bridges in the implementation, except for package imports.

### 9.1.2 Constant

supertype:

System and domain wide constants are mapped to *public static final* member variables of the System of Domain classes. The constants are defined in the initialization hook .pal file along with the value.

### 9.1.3 DataType

supertype: none; subtypes: BaseType, GroupType, GroupIterType, InstanceReferenceType, ServiceHandle, UserDefinedType; The data type for an atomic data item

#### *9.1.3.1 BaseType*

supertype: DataType; subtypes: none; A built-in, predefined type

All base data types map to base data types in Java, int, String, double, etc.

Since Java passes everything by value, it is not possible to support output parameters from services with native Java data types. See the section on Services for more information.

### 9.1.4 UserDefinedType

supertype: DataType; subtypes: UserEnumerate, UserNonEnumerate; A data type defined by the user

#### *9.1.4.1 UserEnumerate*

supertype: UserDefinedType; subtypes: none; An enumerate defined by the user

Enumerated data types are not supported in Java. Instead, they are mapped to a series of *public static final int* member variables of either the System or Domain class, depending upon the scope of their definition.

Another possible implementation to consider here is to make each enumerated type its own Java class. This allows the language's type checking to verify that any use of an enumerate is valid within that context. The class itself contains factory methods to return the human-readable name of the enumerate, the next enumeration in the list, etc.

#### *9.1.4.2 UserNonEnumerate*

supertype: UserDefinedType; subtypes: none; A non-enumerate data type defined by the user

Since there is no concept of a typedef in Java, all non-enumerated user defined types are implemented as their corresponding Java base types.

All realized types will be handles to classes. The base type handle will be used for these types.

In the EJB implementation, the data type must be Serializable, or must be transient everywhere it is used.

### 9.1.5 Domain

supertype: DataTypeScope; subtypes: none

Domains are mapped into Java packages that contain the classes within the domain. In addition, a domain Java class is created that contains the domain services and the domain data types.

In EJB, the domain class will be mapped to a session bean if it has the EjbSessionBean property set to TRUE. The bean may be stateless or stateful, depending on results of further performance and throughput investigations. A property called EJBSessionBeanType could also be defined if the translation templates needed to support both session bean types.

Each EJB session bean that corresponds to a domain will have an instance of a PfdTask object to manage the event queue and timers for services that are invoked through it. The task will execute until there are no more events in the queue and no timers left active. Since the bean is stateless, the PfdTask instance will be refreshed to its initial state of an empty queue when finished with the service.

The domain's session will serve as a wrapper for the domain's implementation in a non-EJB class. The implementation class will look much like it does in the straight Java design, with static methods. Within a process, all services invoked in a domain local to the process will be through the implementation class. Across processors or processes, a session bean will be used. This will reduce the incidence of loopback/reentrance issues, though cross-process loopbacks are still possible.

For Example
```
class DomSessionBean extends SessionBean
{
public PfdTask task = new PfdTask();
void Service()
{ task.clear(); DomImpl.Service(); task.processOOA(); return;};
void Another () { ... };
}

class DomImpl
{
public Service()
{
// regular generates, etc
DomImpl.Another();  // local service

// For cross process, create session bean + invoke session service
...
}
public void Another() { ... };
}
```

OReilly EJB, p362 recommends avoiding chained stateful session beans. Chained session beans are beans that invoke other beans, all stateful. If a session times out, the whole state of that set of beans is corrupted and irrecoverable. Since domain invocations will result in chained session beans, the recommendation seems to be stateless sessions.

### 9.1.6 System

supertype: DataTypeScope; subtypes: none

The system is mapped to a package with the same name as the domain chart. This package contains all of the domain packages and the Sys class, which defines the system wide enumerated types and support for the mechanisms.

## *9.2 Information Modeling Constructs*

### 9.2.1 Attribute

supertype: none; subtypes: none

An Attribute maps to a class attribute in Java. Currently the attribute is package visible and accessed directly in the implementation.

In EJB entity beans, attributes also map to a property or field in persistent classes. All attributes that are to be container managed must be declared public to allow the EJB container to manage them. Read and Write accessors will be implemented using the set/get<attrName> pattern in the bean interface. Since it is possible that some attributes may not be persistent, an optimization would be an Attribute level boolean Transient property.

If the IM is the basis for the RDB schema, using the relational database schema translation templates, then the mappings for EJB can be easily generated. If another schema is used, the EJB mapping becomes more complicated, and should be done using the EJB servers deployment capabilities.

Attributes of container managed entity beans are mapped to database tables through deployment descriptors, which are in XML format for EJB 1.1. Given straightforward mappings, we can generate most of this XML deployment descriptor directly.

### 9.2.2 UML Class

supertype: none; subtypes: none

A UML class is mapped to a Java class. All MBSE modeled classes are subtypes of PfdObject, for passive classes, or PfdActiveObject, for classes with state machines. PfdActiveObject also inherits from PfdObject.

In EJB, the PfdObject and PfdActiveObject are interfaces that are implemented by the EntityBeans.

A class is persistent if the Persistent property is set to TRUE. The translation rules will assume that a table has been created via the RDB templates to correspond to this class.

Persistent classes are mapped into EJB entity beans. Entity beans can be either container managed or bean managed. By default, the translation rules will use container managed persistence, since this is generally easier, and since there is a 1:1 mapping between the classes and the RDB schema. If container managed persistence is required, an EJBEntityBeanType property will be added.

MBSE/UML classes mapped to entity beans will generate 4 classes/interfaces within the domain package:
Remote interface <classname>
Primary Key class <classname>PK
Home interface <classname>Home
Bean class <classname>Bean (Container or bean managed)

See documentation on EJB Entity beans for definitions and uses of these classes and interfaces.

UML classes that participate in sub/super relationships cannot be mapped to a container managed Entity bean. Based on research and prototyping, container management of the superclass bean data does not operate correctly. Bean managed persistence will be used for classes in a sub/super relationship in the analysis.

The remote interface for translated classes will need to integrate with the mechanisms. Interfaces equivalent to PfdObject and PfdActiveObject classes will have to be developed and added to the mechanisms. Attributes of these classes will be added during translation to generated bean classes, as will implementation of the interface methods.

### 9.2.3 Relationship

supertype: none; subtypes: BinaryRel, SubSuperRel

#### *9.2.3.1 BinaryRel*

supertype: Relationship; subtypes:none

By default, a binary association maps to a Java object reference. This assumes that all instances of all classes within the domain are in memory at the same time.

Since, in an EJB implementation, all instances related to an object may not be in memory at once, there must be a way to traverse an association from an in memory class to related classes in the database. To achieve this, formalizing attributes, already added to the tables through the RDB schema translation, will be added to the EJB classes. See the action language section on Relationships for more information on the use of formalizers to traverse relationships, link, and unlink.

References to transient classes will not have a formalizing attribute.

Associations that are [0|1]..* to [0|1]..* (many to many) do not require an associative object in MBSE. The associative table is created by the RDB templates, but no corresponding Java/EJB class will exist.

#### *9.2.3.2 SubSuperRel*

supertype: Relationship; subtypes:none

Subtype or inheritance relationships are mapped to the *extends* keyword in Java. Java does not support multiple inheritance, but then neither does MBSE/UML.

EJB Entity beans implement the EJB interfaces, and can therefore extend other modeled classes or PfdObject or PfdActiveObject.

As mentioned in the Object section, classes in sub/super relationships must be bean managed entity beans,

## *9.3 State and Service Modeling Constructs*

The combination of domain services, state machines and actions, and class services make up the dynamic model of the domain. These constructs model the dynamics of the business logic of the particular domain. In a typical EJB design, all business logic is given to the session beans, keeping the entity beans very simple containers of data and relationships.

So, two basic implementation alternatives exist for mapping the state actions and instance base class services: attach them to the entity bean or create a separate class for them. Since the instance services and state actions cannot be called from outside the domain, there is no need for them to be session beans. The class consisting of state and instance services will be an EJB client and an adjunct to the entity bean. This is in keeping with the EJB model of keeping entity beans as very simple data stores.

### 9.3.1 Action

supertype: none; subtypes: Service, State; the parent of all analysis elements with action language

#### *9.3.1.1 Service*

supertype: Action; subtypes: DomainService, ObjectService

##### 9.3.1.1.1 DomainService

supertype: Service; subtypes: none

For the straight Java design, domain services map to static methods of the domain class. For realized domains, an interface with the same methods and signatures is defined. To realize the interface, this domain interface must be satisfied by a hand-coded class. At startup, an instance of this class is created and registered with the realized domain. The static methods invoked by the analyzed services are then passed to the registered class that implements the domain interface.

For the EJB design, domain services will be mapped to instance based methods of the domain's session bean class. Services that generate events will create a PfdTask instance to manage the event queue and wait until all events are processed and timers expired before returning.

Instance based services in EJB will be mapped to an adjunct class for entity beans. The adjunct class separates the business logic from the bean accessors.

##### 9.3.1.1.2 ObjectService

supertype: Service; subtypes: none

In the Java design, object services will map to Java class methods, static methods for class based services.

In EJB, the class based services will be promoted to the domain level, but with only package visibility. Instance based services will remain with the entity bean or with its corresponding session bean.

Actually, class based services aren't visible outside of the domain, and can therefor be implemented within the second inner layer of the EJB interface design (the implementation layer).

#### *9.3.1.2 State*

supertype: Action; subtypes: none

In the Java design, state actions are implemented as instance based methods invoked by the state machine mechanisms.

State actions in EJB will be mapped to an adjunct class for entity beans. The adjunct class separates the business logic from the bean accessors.

#### *9.3.1.3 Initialization Hooks*

Initialization hooks are pieces of code that are configured (generated) to run automatically on startup. The hooks can be defined at the system or domain levels.

For stateless session beans, all defined initialization hooks must be run before the service executes.

### 9.3.2 Event

supertype: none; subtypes: none

Events are transient within a domain service invocation. All events will carry with them the reference to the PfdTask on which they are queued, so that subsequent event generates resulting from handling the event will be placed on the same queue.

Events destined for active objects that are also entity beans must carry the destination instance reference as an EJBHandle rather than a Java pointer. This is because the EJB container may choose to passivate (remove from memory) the target entity bean between generation and reception of the event.

### 9.3.3 Generate

supertype: EventAccessor; subtypes: none; an invocation of an event generate accessor

Event generation places events on the event queue within the context of a specific Task, shared among all the beans involved in a particular domain service. See the section on MBSE Semantics, subsection Task for more information on this.

### 9.3.4 State Transition Table

The table describes the state machine in terms of current state, incoming event, and next state.

This table will be stored statically in the active object itself. In the case of entity beans, the transition table will be stored in the same class as the state actions themselves.

### 9.3.5 Output Parameters

Since Java passes everything by value, it is not possible to support output parameters from services with native Java data types. Instead, a set of lightweight carrier classes are defined to wrap the base Java types and allow these classes to carry the output data back to the calling service. This restricts calls to services with output parameters to be standalone calls, not part of a more complex expression like an If condition.

In Java RMI and EJB, passing a carrier object will not be sufficient, as the changes are not propagated back to the caller. The only recourse to this is to pass the results in a wrapper class as the return value for the method in question. A serializable return vector must be used to transport output parameters and return values from server back to client.

A return vector of the service of java.util.Hashtable, using name-value pairs within the hashtable as the output parameters, could be used. This return vector will also carry the return value of the service and any service handles that were invoked on the server and destined for the client. The advantage of this interface is that it would be consistent across all client-server services, but the hashtable interface and name-value pairings suffer from possible run-time mismatches in naming if interfaces change.

Alternatively, a serializable class could be generated to carry the return vector. This is the current design plan. All output parameters would map to public member variables, as would the return value. For realized domains, an interface layer that simplifies the calling interface will be generated. See Realized Interfaces for more information.

A superclass mechanism, PfdReturnVector is defined in the SoftwareMechanisms. This is the superclass to all return vectors. It also carries the set of service handles invoked on the server and bound for the client.

## *9.4 Action Language*

### 9.4.1 AttributeSelection

supertype: Expression; subtypes: none; an expression that reads or writes an attribute value

In both the Java and EJB designs, attribute reads and writes map to invocations of the attribute's get or set method.

### 9.4.2 Create

supertype: ObjectAccessor; subtypes: none; an invocation of an object create accessor

In the Java design, create statement maps to the creation of a new in- memory instance.

In EJB, the create statement of a class that is an entity bean maps to the ejbCreate invocation that creates the instance in the underlying database.

### 9.4.3 CreateServiceHandle

supertype: Statement; subtypes: none; an invocation of this built-in service

The CREATE ServiceHandle AL statement in both Java and EJB maps to the instantiation of a PfdServiceHandle, with the domain and service numbers filled in, along with any parameters.
See the discussion in Mechanisms, Service Handles for more information.

### 9.4.4 Delete

supertype: ObjectAccessor; subtypes: none; an invocation of an object delete accessor

In the Java design, delete statement maps to the removal of references to the in-memory instance, resulting in its availability for garbage collection.

In EJB, the delete statement of a class that is an entity bean maps to the remove invocation that deletes the instance in the underlying database.

### 9.4.5 Expression

supertype: none; subtypes: AttributeSelection, BinaryExpression, Constant, EventAccessor, LocalVariable, ObjectAccessor, ParameterVariable, RelationshipAccessor, ServiceInvocation, UnaryExpression; an invocation of a function and/or something that returns/has a value (as an rvalue), or something that can have it's value set (as an lvalue)

Expression subtypes are mapped into Java.

### 9.4.6 Instance Lookups - Find/Foreach

supertype: ObjectAccessor; subtypes: none; an invocation of an object find accessor

A lookup statement in MBSE may have two sources, a relationship navigation or the class instances, and each source Find may be of two types, a find first/last (FIND), find all (FOREACH), and each Find type may include an optional Where clause for further limitations on the returned instances.

In the straight Java design, lookup statements map to searches on the in memory relationship or instance lists. The instances are iteratively compared with the Where clause.

In the EJB design, for entity beans, not all the class instances or related instances may be in memory, so lookup must map to database searches, using the EJB as an interface. Each type of lookup (class based find and relationship traversal) referenced in the AL of the domain will have an interface in the Home interface as well as the bean class.

Custom FINDS can be implemented as separate methods and use SQL to access the database directly. Matching entities must result in the return of a Collection of Primary Key objects, which the container would translate to instances. If required, we can map class based finds and association traversals directly to SQL.

### 9.4.7 InvokeServiceHandle

supertype: Statement; subtypes: none; a statement that invokes a ServiceHandle

The CALL statement invokes a service handle. If the service referred to by the handle resides in the same process, the handle is decoded and invoked. If the target service resides on an EJB server, a session bean is obtained at the service handle invoked. If the target service is invoked on the EJB server and resides on the EJB client, the service handle is added to the return vector of the service for transport back to the client.

See the discussion in Mechanisms, Service Handles for more information.

### 9.4.8 Link

supertype: RelationshipAccessor; subtypes: none; an invocation of a relationship link accessor

In Java, a link results in bi-directional pointers set in the participating objects (and in an associative object, if necessary).

In EJB, a link does the same thing, since both objects must be in memory to be linked. However, for entity beans, the LINK also sets the formalizing database attribute for the relationship. Links may also result in the creation of associative objects in the database to support many to many relationships.

### 9.4.9 Navigation

supertype: RelationshipAccessor; subtypes: none; an invocation of a relationship navigation accessor

In Java, relationship navigation, from within a FIND or FOREACH statement, is taken from the pointer list within the class.

In EJB, for entity beans and relationships between entity beans, the navigation is mapped to a find/lookup method in the Home interface and bean class to find the associated instances from the database.

### 9.4.10 ServiceInvocation

supertype: Expression; subtypes: none; the invocation of an object or domain service

An invocation of a class method of the domain or UML class, in Java.

In EJB, maps to the appropriate invocation of a service, either in a session bean or an entity bean. In some cases, it may need to create a new session bean before invoking the method.

#### 9.4.11 SubSuperNavigation

supertype: Expression; subtypes: none; a "cast" from a supertype to one of its subtypes

Results in a simple downcast in both Java and EJB designs. NOTE: For EJB 1.1, this is done using the javax.rmi.PortableRemoteObject.narrow() method. See O'Reilly pp 133-135 for exceptions.

#### 9.4.12 Unlink

supertype: RelationshipAccessor; subtypes: none; an invocation of a relationship unlink accessor

The reverse of LINK, in Java, it removes the bi-directional references between the instances.

In EJB, it also removes the formalizing attributes from the database table. Unlink will also remove any associative objects in the database.

## *9.5 MBSE Semantics*

### 9.5.1 Timers

A single timer thread will be shared among all the session beans implementing the domain interfaces. The timer thread will accept timer settings and respond to the requesting Task when the timer has expired. This thread should be started outside the EJB environment, but shareable within it.

### 9.5.2 Instance Lists

An instance list is the set of all instances of a particular class. In this design, the set is only completely visible within the relational database that supports the system. As such, there are really 2 instance lists – the complete set in the database, and the partial set in memory. It is assumed that the EJB container takes care of insuring that an instance (entity bean) is not in memory twice.

Transient class instances are held completely in memory, but should not overlap between tasks or instances of the domain.

No in-memory instance lists are kept for entity beans, since the database keeps the list.

In memory lists for non-entity beans must be kept on a thread-by thread basis. The instance list must then be part of the thread's MBSE execution context.

### 9.5.3 Service Handles

EJB does not provide any support for asynchronous communication between two entities. The model is strictly client server, where all processing is synchronous and controlled by the client, invoking the server and blocking until processing is complete and the results returned. Thus, the concept of service handles does not map well to the communication between domains on the client and domains on the server.

Service handles invoked on the EJB server that are services of the client will be serialized across the network, carried in the return vector for the service. Once they arrive at the client, they are deserialized and invoked. Note that this limits a server thread from communicating with other clients, only the client that started the particular thread.

Domain services of the client will be wrapped in an interface class that will create the service handle and insert it into the return vector. As part of the return vector from the service, a set of zero or more service handles will be returned. The service handles will be serialized to cross the wire. Any service handles bound for the client will be returned in the return vector, unpacked, and invoked before the method returns on the client. The service would also package up the return values to return them to the client.

Between domains on the same side of the client-server boundary, the mapping is straightforward. Either the service handle is decoded and invoked directly or a session bean is created and then the service invoked.

#### 9.5.4 Task

The task consists of the MBSE event loop and the interface to timers. Within the context of the EJB design, there could be many tasks running at the same time, coordinated through the EJB container and the underlying relational database.

Each domain session bean service accepts parameters, does processing, and, if it generates an event, creates a task, deposits the event, and spins the event loop. The event loop executes until there are no events left to process and there are no timers active.

Since there are multiple tasks running at the same time, the objects must know what task to place new events into. There is no global area or global key that can be used to store the PfdTask reference, so it must be passed around during processing. The PfdTask reference is included with each event, so the receiving states will know what PfdTask to enqueue the next event on. Any services that can generate an event will have a PfdTask added to the interface. Session bean interfaces do not expose the PfdTask, but include their PfdTask in calls to domain implementation services.

We may need to add the PfdTask to all services, since a service may call another service that may generate an event. Or, the whole tree of processing for a service may need to be explored during translation.

# 10. Implementation Specific Issues

#### 10.1.1 Exceptions

Exception handling is not part of the MBSE semantics. Java exceptions are not thrown or caught by the mechanisms or any of the design templates.

### *10.2 EJB Specific Issues*

#### 10.2.1 Loopbacks

#### 10.2.2 Exceptions

Exception handling is not part of the MBSE semantics. No application exceptions are not thrown or caught by the mechanisms or any of the design templates. EJB exceptions are caught and rethrown as required by the EJB specification.

#### 10.2.3 Transactions

The EJB container and server provide distributed transaction context for the application. The EJB container provides a set of properties on EJB classes and methods that can be used to control and optimize access to the database while providing enough protection to insure that data remains consistent. Transaction membership and isolation control are the properties that can be set. The level of support for these varies between vendors and must be investigated for TopLink and Oracle.

These EJB properties are set in the deployment descriptor for the bean. A set of properties can be captured in the MBSE models that correspond to the EJB properties. The deployment descriptor can be generated from these.

We assume that helper classes – non EJB classes called from a bean - carry transaction processing along, and that the transaction will propagate to other beans called by the helper classes as per the EJB transaction propagation rules and settings.

Deadlocks are detected by the EJB container or the underlying database when tested with the BEA WebLogic server and the Cloudscape database. In a deadlock situation, one of the session clients will time out, causing the server to roll back any pending transactions.

## *10.3 Realized Interfaces*

The interface between client and service in mapping MBSE to EJB gets a bit complicated with service handles and output parameters, neither of which are directly supported by EJB. This is not a problem for analyzed domains, since the code can be generated to deal with that. With realized domains, such as the user interface, it becomes more of a problem. To ease the realized to analyzed interface across the EJB client-server boundary, a simplified wrapper class around the session bean will be provided.

The adapter class will have the same interface as the MBSE models, with carrier classes for output parameters. The decoding of the return vector and invocation of any service handles will be performed before the service returns.

## 11. Porting

## 12. Modeling Conventions and Restrictions

Springboard expects your UML models follow the conventions described in "Pathfinder Solutions UML Modeling Guide".

Ideally, there are no conventions and restrictions placed on the analysis by the implementation or design technology. Practically, there is a small amount of feedback, especially for a design that is under development and intended to be used before it is complete. The following are conventions and constraints on the model for use with the Java EJB design. This list will change as development progresses.

- Use of output parameters from services in complex expressions. Service calls with output parameters cannot be used within a compound expression, like an IF or WHILE statement.
- Loopback limitation on entity beans and session beans. Make as much inter-class communication as possible asynchronous via events to avoid this.

## 13. TERMS and DEFINITIONS

| | |
|---|---|
| Templates | a set of text files that capture patterns to define the structure, fixed contents, and variable contents (corresponding to UML model elements) of the target document set, using Syntactic Elements that conform to the notation specified in the Springboard User's Guide. |
| Extraction | the portion of the translation process when Springboard reads the model database captured in your editor |
| MBSE Meta-Model | the MBSE/UML class diagram that describes MBSE and stores the application models for translation. |
| Parsing | the portion of the translation process when Springboard reads the templates in preparation for production |
| Production | the portion of the translation process when Springboard executes the templates, populating your target document set with the OOA information acquired during extraction |
| Property | a name-value pair associated with a UML model element |
| Target document | a file or document produced by Springboard as the result of executing an OUTFILE directive |
| Target document set | the complete set of target documents produces by a single execution of Springboard on a set of templates |
| Translation | the execution of Springboard performs 3 steps in order: Template Parsing, model Extraction, and Target Document Production |

## 14. Referenced Documents

Enterprise JavaBeans, Second Edition, Richard Monson-Haefel, OReilly and Associates

Springboard User's Guide

Translation Overview white paper

"Model Based Software Engineering – an Overview of Rigorous and Effective Software Development Using UML™" 1998, Pathfinder Solutions, Inc.

editor specific modeling guides