A Metamodel-Based Model Compiler

Shlaer-Mellor User Group Conference
May 23-24, 2000

Chuck Harvey
Dan Walden
Overview / Topics

- Background
- Recursive design process and model compiler’s role
- Metamodel-based model compiler design
  - Requirements
  - Structure
  - Control
- Future enhancements
- Conclusions and observations
Background

- Why develop a model compiler?
  - Off-the-shelf model compiler not available
  - CORBA / Java implementation environment

- Why develop a metamodel-based model compiler?
  - Approach supports “natural partitioning of work,” understandable
  - Reuse on future projects
  - An eye toward a family of compilers
Recursive Design Process

Characterize the System

Define Conceptual Entities of the Architecture

Define the Theory of Operation

Recursive Design Process - Continued

Build Archetypes

Collect Instance Data

Populate The Architecture

Generate Code
## Model Compiler’s Role

<table>
<thead>
<tr>
<th>RD Process</th>
<th>Model Compiler Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characterize the system</td>
<td>-</td>
</tr>
<tr>
<td>Define conceptual entities</td>
<td>Implementation form (code or data)</td>
</tr>
<tr>
<td>Define theory of operation</td>
<td>-</td>
</tr>
<tr>
<td>Collect instance data</td>
<td>OOA mapping, coloring, allocation</td>
</tr>
<tr>
<td>Populate the architecture</td>
<td>Architecture metamodel population</td>
</tr>
<tr>
<td>Build the archetypes</td>
<td>Archetype structure</td>
</tr>
<tr>
<td>Generate code</td>
<td>Archetype processing</td>
</tr>
</tbody>
</table>
Code Generation Through Translation - Concept

OOA Metamodel → Model Compiler

Archetypes (Templates) → Code Generator → Source Code
Code Generation Through Translation - Metamodel Based

- OOA Metamodell
- Model Translator
- Software Architecture Metamodell
- Source Code Translator
- OOA Entity To Software Architecture Entity Translation Rules
- Software Architecture Entity To Implementation Entity Translation Rules
- Source Code
Model Compiler Requirements and Implementation Environment

- Project tool, not a commercial compiler
- Two target languages: CORBA Interface Definition Language (IDL) and Java
- OOA Models captured using BridgePoint visual modeling tool (Version 4.0)
  - OOA metamodel
  - Archetype language
- Project Technology DesignPoint TinyJava BPAL components
Model Compiler Subsystem Relationship Diagram

<<Subsystem>>

OOA-Metamodell

<<Subsystem>>

OOA-Architecture Mapping

<<Subsystem>>

OOA Coloring

<<Subsystem>>

Software Architecture Metamodell

<<Subsystem>>

Compiler Control
OOA-Architecture Mapping

Subsystem

- Representative entities:
  - Domain - Domain mapping
  - Subsystem - Subsystem mapping
  - Object-Domain object mapping
  - Attribute - Attribute mapping
  - State model - Finite state machine mapping
  - State model event - Event mapping
  - State model event data item - Event data item mapping
  - External entity - External entity mapping
OOA Coloring Subsystem

- Representative entities:
  - Domain coloring (parent package)
  - Object coloring (pooled, initial pool size, distributed, replicated, adapter, conforming)
  - Attribute coloring (color, size, precision, scale, array)
  - Event coloring (pooled, initial pool size)
  - State coloring (logged in final state)
  - External entity coloring (conforming, server package, server adapter class)
Software Architecture Subsystem

- Representative entities:
  - Data type (native, constructed, object reference, enumerated)
  - Package
  - Domain object (passive, active, adapter, initialization)
  - Finite state machine
  - Object state
  - Domain event
  - Event
  - Supplemental data set
  - External entity adapter
Model Compilation Process

- Initialization
- Population of OOA metamodel, including BPAL
- Population of software architecture metamodel
  - Software architecture subsystem data types
  - Mapping and coloring subsystems
  - Remaining architecture subsystem entities
- Production of source code
  - Java, including translation of BPAL actions
  - CORBA Interface Definition Language
  - Make scripts
- Cleanup and gathering of statistics
State Model State Subset

Mapping Subsystem

Software Architecture Metamodel

OOA Metamodel

Coloring Subsystem

Cambridge Technology Partners
Example Object Information Model
Example State Model

1. **Inoperable**
   
   entry/
   Generate dp_L2 to self;

   dp_L1: `Lamp_inserted`
   (Color, Label, Mode)

   dp_L2: `Turn_off`

   dp_L2: `Turn_off`

2. **OFF**
   
   entry/
   Generate dp_L3 (Mode: "Steady") to self;

   dp_L4: `Remove`

3. **ON**
   
   entry/
   Generate dp_L2 to self;

   dp_L3: `Turn_on`

   dp_L4: `Remove`

4. **Service**
Preexisting Instance Population
Populated Example Preexisting Instance Object

**State**

<table>
<thead>
<tr>
<th>State_ID</th>
<th>Name</th>
<th>Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>_UNDEFINED</td>
<td>0</td>
<td>SystemDefinedState</td>
</tr>
<tr>
<td>99</td>
<td>_DONOTCARE</td>
<td>-1</td>
<td>SystemDefinedState</td>
</tr>
<tr>
<td>100</td>
<td>_CANNOTHAPPEN</td>
<td>-2</td>
<td>SystemDefinedState</td>
</tr>
<tr>
<td>101</td>
<td>Inoperable</td>
<td>1</td>
<td>ModeledState</td>
</tr>
<tr>
<td>102</td>
<td>OFF</td>
<td>2</td>
<td>ModeledState</td>
</tr>
<tr>
<td>103</td>
<td>ON</td>
<td>3</td>
<td>ModeledState</td>
</tr>
<tr>
<td>104</td>
<td>Service</td>
<td>4</td>
<td>ModeledState</td>
</tr>
</tbody>
</table>

**PredefinedState**

<table>
<thead>
<tr>
<th>Predefined State_ID</th>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_UNDEFINED</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>_DONOTCARE</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>_CANNOTHAPPEN</td>
<td>-2</td>
</tr>
</tbody>
</table>

**ModeledState**

<table>
<thead>
<tr>
<th>State_ID</th>
<th>isLoggedInFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>False</td>
</tr>
<tr>
<td>102</td>
<td>False</td>
</tr>
<tr>
<td>103</td>
<td>False</td>
</tr>
<tr>
<td>104</td>
<td>True</td>
</tr>
</tbody>
</table>

**SystemDefinedState**

<table>
<thead>
<tr>
<th>State_ID</th>
<th>Predefined State_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
</tr>
</tbody>
</table>
Mapping And Coloring
Populated Example Mapping and Coloring Objects

State Model State

<table>
<thead>
<tr>
<th>SMstt_ID</th>
<th>Name</th>
<th>Numb</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Inoperable</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>OFF</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>ON</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Service</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

StateModelState

<table>
<thead>
<tr>
<th>SMstt_ID</th>
<th>State_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>101</td>
</tr>
<tr>
<td>1002</td>
<td>102</td>
</tr>
<tr>
<td>1003</td>
<td>103</td>
</tr>
<tr>
<td>1004</td>
<td>104</td>
</tr>
</tbody>
</table>

StateColoring

<table>
<thead>
<tr>
<th>SMstt_ID</th>
<th>isLoggedInFinal</th>
<th>State_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1004</td>
<td>True</td>
<td>104</td>
</tr>
</tbody>
</table>

ModeledState

<table>
<thead>
<tr>
<th>State_ID</th>
<th>isLoggedInFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>False</td>
</tr>
<tr>
<td>102</td>
<td>False</td>
</tr>
<tr>
<td>103</td>
<td>False</td>
</tr>
<tr>
<td>104</td>
<td>True</td>
</tr>
</tbody>
</table>
Software Architecture Metamodel Population
## Populated Example Software Architecture Objects

### State

<table>
<thead>
<tr>
<th>State_ID</th>
<th>Name</th>
<th>Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>_UNDEFINED</td>
<td>0</td>
<td>SystemDefinedState</td>
</tr>
<tr>
<td>99</td>
<td>_DONOTCARE</td>
<td>-1</td>
<td>SystemDefinedState</td>
</tr>
<tr>
<td>100</td>
<td>_CANNOTHAPPEN</td>
<td>-2</td>
<td>SystemDefinedState</td>
</tr>
<tr>
<td>101</td>
<td>Inoperable</td>
<td>1</td>
<td>ModeledState</td>
</tr>
<tr>
<td>102</td>
<td>OFF</td>
<td>2</td>
<td>ModeledState</td>
</tr>
<tr>
<td>103</td>
<td>ON</td>
<td>3</td>
<td>ModeledState</td>
</tr>
<tr>
<td>104</td>
<td>Service</td>
<td>4</td>
<td>ModeledState</td>
</tr>
</tbody>
</table>

### PredefinedState

<table>
<thead>
<tr>
<th>PredefinedState</th>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_UNDEFINED</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>_DONOTCARE</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>_CANNOTHAPPEN</td>
<td>-2</td>
</tr>
</tbody>
</table>

### ModeledState

<table>
<thead>
<tr>
<th>State_ID</th>
<th>isLoggedInFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>False</td>
</tr>
<tr>
<td>102</td>
<td>False</td>
</tr>
<tr>
<td>103</td>
<td>False</td>
</tr>
<tr>
<td>104</td>
<td>True</td>
</tr>
</tbody>
</table>

### SystemDefinedState

<table>
<thead>
<tr>
<th>State_ID</th>
<th>Predefined State_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
</tr>
</tbody>
</table>
Populator Archetype Design

```vbnet
//
// Replace {Object} with Software Architecture Entity Name
//=======================================================================
// Name: Add {Object}s
// Desc: Process the set of Entities
//=======================================================================
.Function Add{Object}s
.Param inst_ref ooa_parent_entity
// Any required Params for formalization (above this line).
.Select many ooa_entityS related by ooa_parent_entity->OOA XXX[Rn]
.For each ooa_entity in ooa_entityS
   // Process Coloring Data (if applicable)
   // Get Unique ID
   .Invoke self = GetUniqueId()
   // Add self
   .Invoke sa_entity = Add{Object} (ooa_entity, self, (parms...) )
   ${sa_entity.body}
   // Add Mapping
   .Invoke map = Add{Object} Mapping(ooa_entity, self, (parms...) )
   ${map.body}
   // Handle Associative Objects ( as needed )
   // process next model element(s)
   .Invoke next = Add{ Next Model Element}(parms)
   ${next.body}
 .End For
 .End Function
```
Generated Example Java Source

```java
final class Lamp_STATE extends sa_ObjectState {
    public static final int Inoperable = 1;
    public static final int OFF = 2;
    public static final int ON = 3;
    public static final int Service = 4;
}
```
Future Enhancements

- Move to BridgePoint Version 5
- Extend metamodel to include action implementation entities
- Add metamodel(s) for implementation domains
- Investigate the formalization of concept using abstract syntax trees
Conclusions And Observations

- It can be done!
- Increased modularity
- Increased maintainability
Thank You!

Questions, Anyone