BridgePoint® - Automation

Release 3.3 to 4.0

BridgePoint® Customer Support

http://www.projtech.com/support/bpcsa.html

User ID___________________ Password__________________

Tel: 800-482-3853 or 520-544-0808
email: support@projtech.com

Automation
Use, examination, reproduction, copying, transfer, and/or disclosure of
“BridgePoint® - Automation”
to others is prohibited except by express agreement with
Project Technology, Inc.

Copyright © 1992-1999
Project Technology, Inc. and its licensors.
7400 N. Oracle Road, Suite 365 Tucson, AZ 85704-6342 USA
All rights reserved.

The Database Management portion of this product is based on:
ObjectStore®
Copyright © Object Design, Inc. 1988-1998
All rights reserved. Patent Pending.

The License Management portion of this product is based on:
Elan License Manager
Copyright © Rainbow Technologies, Inc. 1998
All rights reserved.

All other products or services mentioned in this document are identified by the trademarks, service
marks, or product names as designated by the companies who market those products.

RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the Government is subject to restrictions as set forth in
subparagraphs (c)(1)(ii) of the
Rights in Technical Data and Computer Software
clause at 252.227-7013 (48 CFR, Ch.2).
Contractors/manufacturers are
Project Technology, Inc., 7400 N. Oracle Rd., Ste. 365, Tucson, AZ 85704-6342 USA
Object Design, Inc., 25 Burlington Mall Road, Burlington, MA 01803 USA
Rainbow Technologies, Inc. USA

Version 4.0-1.3
This manual is part of the three manual set for the BridgePoint family. Here is an overview of the three manuals.

**BridgePoint - OOA**
This manual is built around the process of doing OOA. The OOA process is broken down into steps and each step has Process, Method, and Automation sections in the manual. The Process section addresses who does the step, how long it should take, the outputs of the step, and quality issues. The Method section describes the underlying method issues for this step. The Automation section describes how the BridgePoint tool set is used to automate the OOA step.

**BridgePoint - Automation**
This manual is built around the process of doing file generation using Design By Translation techniques using the BridgePoint Generator tools.

**BridgePoint - Tool Guide**
This manual deals with BridgePoint tool specific topics that are independent of OOA. Included in this manual are Tool Fundamentals, Specifying User Properties,
General Model Editing, Printing, Version Management, License Management, Importing and Exporting Data, and BridgePoint Installation.
# Table of Contents

<table>
<thead>
<tr>
<th>TASK</th>
<th>Architecture Blueprint</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEPSTEP 7</td>
<td>Architecture Characterization</td>
<td>3</td>
</tr>
<tr>
<td>7.1 Method</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>7.1.1 Implementation Technologies</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>7.1.2 Architectural Decisions</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>STEPSTEP 8</td>
<td>Architecture Design</td>
<td>13</td>
</tr>
<tr>
<td>8.2 Method</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>8.2.1 Subsystem ‘Subsystem’</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>8.2.2 Subsystem ‘Object’</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>8.2.3 Subsystem ‘Relationship’</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>8.2.4 Subsystem ‘Communication &amp; Access’</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>8.2.5 Subsystem ‘State Model’</td>
<td></td>
<td>97</td>
</tr>
</tbody>
</table>
## TASK SW Arch Implementation

### STEP STEP 9 Develop Structural Archetypes

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Method</td>
<td>125</td>
</tr>
<tr>
<td>9.1.1 General Language Attributes</td>
<td>126</td>
</tr>
<tr>
<td>9.1.2 Literal Text</td>
<td>127</td>
</tr>
<tr>
<td>9.1.3 Data Access Control Statements</td>
<td>128</td>
</tr>
<tr>
<td>9.1.4 Transformer Control Statements</td>
<td>135</td>
</tr>
<tr>
<td>9.1.5 Tester Control Statements</td>
<td>137</td>
</tr>
<tr>
<td>9.1.6 Function Control Statements</td>
<td>138</td>
</tr>
<tr>
<td>9.1.7 File Control Statements</td>
<td>141</td>
</tr>
<tr>
<td>9.1.8 Rvalues</td>
<td>143</td>
</tr>
<tr>
<td>9.1.9 Expressions</td>
<td>145</td>
</tr>
<tr>
<td>9.1.10 Substitution Variables</td>
<td>151</td>
</tr>
<tr>
<td>9.2 Automation</td>
<td>155</td>
</tr>
<tr>
<td>9.2.1 Overview</td>
<td>155</td>
</tr>
<tr>
<td>9.2.2 Running gen_import</td>
<td>156</td>
</tr>
<tr>
<td>9.2.3 Running gen_file</td>
<td>156</td>
</tr>
<tr>
<td>9.2.4 Using Makefiles</td>
<td>157</td>
</tr>
</tbody>
</table>

### STEP STEP 10 Develop Action Archetypes

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1 Method</td>
<td>161</td>
</tr>
<tr>
<td>10.1.1 Invoking Fragment Generation</td>
<td>163</td>
</tr>
<tr>
<td>10.1.2 Fragment Generation Script</td>
<td>164</td>
</tr>
<tr>
<td>10.1.3 Fragment Generation Functions</td>
<td>166</td>
</tr>
<tr>
<td>10.2 Automation</td>
<td>173</td>
</tr>
<tr>
<td>10.2.1 Overview</td>
<td>173</td>
</tr>
</tbody>
</table>
TASK

Architecture Blueprint
STEP 7

Architecture Characterization
7.1 Method

The Software Architecture Blueprint is a detailed plan of how to make use of a set of Implementation Technologies to implement the functionality defined by the OOA.

The following figure shows what a Software Architecture Blueprint includes:

The following sections outline a set of questions which will guide the determination of the Software Architecture Blueprint. The questions are based upon our experience and consulting - the list is in no way complete and we invite additions to the lists based on your experiences.
7.1.1 Implementation Technologies

7.1.1.1 Programming Language:
- C
- C++
- Smalltalk
- ADA
- COBOL
- Fortran
- Object COBOL
- Assembler
- Objective C
- ...

7.1.1.2 Operating System:
- Mainframe:
  - IBM
  - ...
- UNIX:
  - SunOS
  - System V Release 4
  - HP-UX
  - IRIX
  - AIX
  - ...
- Lightweight Tasking:
  - VxWorks
  - pSOS
  - ...
- OS/2
• Windows
• Windows NT
• ...

7.1.1.3 **Database:**
• Custom
• Relational Database:
  • Oracle
  • Sybase
  • Informix
  • ...
• Object-Oriented Database:
  • ObjectStore
  • Gemstone
  • Objectivity
  • Versant
  • ...

7.1.1.4 **User Interface:**
• Character Based:
  • Custom
  • Off-the-Shelf:
  • XVT
  • ...
• Graphical User Interface (GUI):
  • Custom
  • Off-the-Shelf:
    • Galaxy
    • Motif - COSE
    • MS Windows
    • XVT
    • Openlook - XView
  • ...

7.1.1.5 Implemented Service Domains:
• Alarming
• Measurements
• Audits
• Initialization
• ...

7.1.2 Architectural Decisions

7.1.2.1 Data Organization:
• Data Location:
  • Centralized
  • Distributed:
    • Instance Consolidated
    • Instance Dispersed
Logical Organization:
- One Table/Class per OOA Object
- One or More Table/Class per OOA Object
- 1:1 Objects Aggregated
- 1:M Objects Aggregated (must use design info to specify what M)

Physical Organization:
- Direct correspondence to Logical Organization
- Static Attributes stored separately from Dynamic Attributes (allows direct disk to memory pump of static data image)

Memory Management:
- Operating System, e.g., use default new operator in C++
- Array/Pool - build own specialized instance management mechanisms

Instance Relationships:
- Object-Oriented Approach: Store Related Instances’ Handles - direct access to related instances:
  - Store handles as Table Offsets
  - Store handles as Pointers
- Relational-Oriented Approach: Store Referential Attributes (foreign keys) and search for related instances:
  - Linear Search
  - Hashing
  - B-Tree

Relationship Integrity:
- Require all Unconditional Relationships to be instantiated upon object instance creation
- Allow temporary states of Relationship Conditionality inconsistency

Referential Attributes:
- Store as part of Physical Organization
- Look up across relationship when accessed

Derived Attributes:
- Store derived value - re-derive value when dependent data changes
- Derive for each access
7.1.2.2 Control Organization:

- Action Organization:
  - State Model Consolidated
  - Thread Consolidated
  - Task Allocation:
    - Single Task
    - Multi-Task
      - Peer-Peer
        - Homogeneous
        - Heterogeneous
      - Client-Server
        - Homogeneous
        - Heterogeneous

- Event Priority
  - Events Processed in order Generated
  - Intra-State Model Events Processed first, all other events processed in order Generated
  - Intra-State Model Events Processed first, Inter-State Model Events Processed next, all other events processed in order Generated
• Event Communication Mechanisms:
  • Intra-State Model Events:
    • 1 Queue
    • 1 Queue per Task
    • 1 Queue per State Model
    • 1 Queue per State Machine
    • Function Call
  • Inter-State Model Events:
    • 1 Queue
    • 1 Queue per Task
    • 1 Queue per State Model
    • 1 Queue per State Machine
    • Function Call
  • Inward Bound State Model Events:
    • 1 Queue
    • 1 Queue per Task
    • 1 Queue per State Model
    • 1 Queue per State Machine
    • Function Call
  • Outward Bound External Entity Events:
    • 1 Queue
    • 1 Queue per Task
    • 1 Queue per External Entity
    • Function Call
• Timers:
  • Polling
  • Invocation upon Expiration

7.1.2.3 Source Code Organization:
Step 7: Architecture Characterization

- Code Naming:
  - Key-letters
  - Names
  - Arbitrary IDs
- Code Allocation to Files:
  - Header Files:
    - 1 File per Domain
    - 1 File per Subsystem
    - 1 File per Object
  - Source Files:
    - 1 File per Domain
    - 1 File per Subsystem
    - 1 File per Object
STEP 8

Architecture Design
8.2 Method

This Domain is modeling the semantic data items which make up OOA. Note that this does NOT include the graphical data associated with the models of OOA - that information is captured in a separate domain.

Completeness of this domain is very important - this OOA is to be used to enable translation, i.e. source code generation, from an analysis.
8.2.1 **Subsystem ‘Subsystem’**

A Subsystem is based on the partitioning of an entire Domain. The number of Subsystems in a Domain is dependent upon the Domain subject matter and complexity.

A Subsystem is composed of objects that tend to cluster, i.e., they have many interconnections with one another but few interconnections with objects in different clusters.

Inter-Subsystem relationships, communications, and accesses are captured in the Subsystem Relationship Model (SRM), Subsystem Communication Model (SCM), and Subsystem Access Model (SAM) respectively.
Step 8: OOA of OOA

1. Domain (S_DOM)
   - Dom_ID
   - Name
   - Descrp
   - Full_Der
   - Config_ID
   is first level of partitioning for

2. Subsystem (S_SS)
   - SS_ID
   - Name
   - Descrp
   - Prefix
   - Num_Rng
   - Dom_ID (R1)
   is partitioned into

3. External Entity (S_EE)
   - EE_ID
   - Name
   - Descrp
   - Key_Lett
   - Dom_ID (R8)
   interacts with

4. External Entity Data Item (S_EEDI)
   - EEdi_ID
   - EE_ID (R11)
   - Name
   - Descrp
   - DT_ID (R15)
   can be accessed synchronously via

5. Data Type (S_DT)
   - DT_ID
   - Dom_ID (R14)
   - Name
   - Descrp
   contains defined

6. Core Data Type (S_CDT)
   - DT_ID (R17)
   - Core_Typ
   contains defined

7. Bridge (S_BRG)
   - Brg_ID
   - EE_ID (R19)
   - Name
   - Descrp
   - Brg_Typ
   - DT_ID (R20)
   contains defined domain of

8. User Data Type (S_UDT)
   - DT_ID (R17)
   - Core_DT_ID (R18)
   - User_Typ
   contains defined domain of

9. Bridge Parameter (S_BPARM)
   - BParm_ID
   - Brg_ID (R21)
   - Name
   - Descrp
   - Full_Der
   - DT_ID (R22)
   contains defined domain of

10. External Entity Data Item (S_EDT)
    - EDT_ID
    - EE_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    can be accessed synchronously via

11. Bridge Entity Data Item (S_BEDI)
    - BEdi_ID
    - EEdi_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    can be accessed synchronously via

12. Bridge Parameter (S_BPARM)
    - BParm_ID
    - Brg_ID (R21)
    - Name
    - Descrp
    - Full_Der
    - DT_ID (R22)
    contains defined domain of

13. Bridge Entity Data Item (S_BEDI)
    - BEdi_ID
    - EEdi_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    can be accessed synchronously via

14. Domain (S_DOM)
    - Dom_ID
    - Name
    - Descrp
    - Full_Der
    - Config_ID
    defines type within

15. Data Type (S_DT)
    - DT_ID
    - Dom_ID (R14)
    - Name
    - Descrp
    - Full_Der
    - Config_ID
    defines type within

16. Bridge (S_BRG)
    - Brg_ID
    - EE_ID (R19)
    - Name
    - Descrp
    - Brg_Typ
    - DT_ID (R20)
    defines the return value

17. Bridge Parameter (S_BPARM)
    - BParm_ID
    - Brg_ID (R21)
    - Name
    - Descrp
    - Full_Der
    - DT_ID (R22)
    defines the return value

18. Bridge Entity Data Item (S_BEDI)
    - BEdi_ID
    - EEdi_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    defines the type of

19. Bridge Parameter (S_BPARM)
    - BParm_ID
    - Brg_ID (R21)
    - Name
    - Descrp
    - Full_Der
    - DT_ID (R22)
    defines the type of

20. Bridge Entity Data Item (S_BEDI)
    - BEdi_ID
    - EEdi_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    defines the type of

21. Bridge (S_BRG)
    - Brg_ID
    - EE_ID (R19)
    - Name
    - Descrp
    - Brg_Typ
    - DT_ID (R20)
    defines the return value

22. Bridge Parameter (S_BPARM)
    - BParm_ID
    - Brg_ID (R21)
    - Name
    - Descrp
    - Full_Der
    - DT_ID (R22)
    defines the return value

23. Bridge Entity Data Item (S_BEDI)
    - BEdi_ID
    - EEdi_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    defines the type of

24. Bridge (S_BRG)
    - Brg_ID
    - EE_ID (R19)
    - Name
    - Descrp
    - Brg_Typ
    - DT_ID (R20)
    defines the return value

25. Bridge Parameter (S_BPARM)
    - BParm_ID
    - Brg_ID (R21)
    - Name
    - Descrp
    - Full_Der
    - DT_ID (R22)
    defines the return value

26. Bridge Entity Data Item (S_BEDI)
    - BEdi_ID
    - EEdi_ID (R11)
    - Name
    - Descrp
    - DT_ID (R15)
    defines the type of
8.2.1.1 Object and Attribute Descriptions

1. Domain (S_DOM)

Domain (Dom_ID, Name, Descrip, Full_Der, Config_ID)
Identifier *: Dom_ID
Description: A typical software system generally consists of distinct and independent subject matters. A Shlaer/Mellor analysis partition is based within each of these subject matters - each subject matter is called a Domain. A Domain is inhabited by its own conceptual entities (called objects). A domain may be partitioned into subsystems depending upon its complexity. Each Domain is given a mission statement which provides a charter for the construction of the OOA models.

Domain.Dom_ID
Full Name: Domain Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Domain.Name
Full Name: Domain Name
Attribute Type: Base Attribute
Data Domain: string

Domain.Descrip
Full Name: Domain Description
Attribute Type: Base Attribute
Data Domain: string

Domain.Full_Der
Full Name: Fully Derived Flag
Attribute Type: Base Attribute
Data Domain: boolean
Method

Description: A flag indicating whether the Object Communication Model and Object Access Model are fully derived from the information contained in the Object Information Model and Action Specifications.

Value 0 indicates OCM and OAM are not fully derived.
Value 1 indicates OCM and OAM are fully derived.

Domain.Config_ID

Full Name: Configuration Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Description: The Configuration ID of the version management configuration which the domain is a part of (See Page 57 of BridgePoint Tool Guide). This ID can be used to access the V_CONFIG record corresponding to the Domain/Subsystem Configuration.

2. Subsystem (S_SS)

Subsystem (SS_ID, Name, Descrip, Prefix, Num_Rng, Dom_ID)

Identifier *: SS_ID

Description: A Subsystem is based on the partitioning of an entire Domain. The number of Subsystems in a Domain is dependent upon the Domain subject matter and complexity.

A Subsystem is composed of objects which tend to cluster, i.e., objects which have many relationships with one another but few relationships with objects in different clusters.

Inter-Subsystem relationships, asynchronous communications, and synchronous accesses are captured in the Subsystem Relationship Model, Subsystem Communication Model and Subsystem Access Model, respectively.

Subsystem.SS_ID

Full Name: Subsystem Identifier
Attribute Type: Base Attribute
Data Domain: unique_id
Subsystem.Name
Full Name: Subsystem Name
Attribute Type: Base Attribute
Data Domain: string

Subsystem.Descrip
Full Name: Subsystem Description
Attribute Type: Base Attribute
Data Domain: string

Subsystem.Prefix
Full Name: Subsystem Keyletter Prefix
Attribute Type: Base Attribute
Data Domain: string
Description: The subsystem keyletter prefix is used when objects are created in the subsystem - the subsystem keyletter prefix is used as the default prefix in the object keyletters.

Subsystem.Num_Rng
Full Name: Subsystem Number Range Start
Attribute Type: Base Attribute
Data Domain: integer
Description: The subsystem number range start is used when objects and relationships are created in the subsystem - the subsystem number range start is used as the default auto-numbering start value in for the newly created Object’s number and newly created Relationship’s number.

Subsystem.Dom_ID
Attribute Type: Referential Attribute
Refers To: Domain.Dom_ID (R1) (See Page 20)

3. External Entity (S_EE)
External Entity (EE_ID, Name, Descrip, Key_Lett, Dom_ID)
Identifier *: EE_ID
Description: An External Entity represents something outside of the Domain being modeled that interacts with objects within the Domain being modeled. The interactions are showed by Event Communications in the Object Communication Models and Data Accesses in the Object Access Models. Each External Entity is given a unique name and keyletters within a Domain.

**External Entity.EE_ID**

Full Name: External Entity Identifier  
Attribute Type: Base Attribute  
Data Domain: `unique_id`

**External Entity.Name**

Full Name: External Entity Name  
Attribute Type: Base Attribute  
Data Domain: `string`

**External Entity.Descrip**

Full Name: External Entity Description  
Attribute Type: Base Attribute  
Data Domain: `string`

**External Entity.Key_Lett**

Full Name: External Entity Key Letters  
Attribute Type: Base Attribute  
Data Domain: `string`

**External Entity.Dom_ID**

Attribute Type: Referential Attribute  
Refers To: Domain.Dom_ID (R8)  
(See Page 20)

---

### 4. External Entity in Model (S_EEM)

External Entity in Model (EEmod_ID, EE_ID, Modl_Typ, SS_ID)  
Identifier *: EE_ID, EEmod_ID
Step 8: OOA of OOA

Description: The External Entity in Model is the presence of an External Entity in a model such as the Object Communication Model or Object Access Model. The same External Entity can be represented by more than one External Entity in Model in the same model to enhance model layout.

**External Entity in Model.EEmod_ID**
- Full Name: External Entity in Model Identifier
- Attribute Type: Base Attribute
- Data Domain: unique_id

**External Entity in Model.EE_ID**
- Attribute Type: Referential Attribute
- Refers To: External Entity.EE_ID (R9) (See Page 23)

**External Entity in Model.Modl_Typ**
- Full Name: Model Type
- Attribute Type: Base Attribute
- Data Domain: integer
- Description: Value indicates what type of model the External Entity is in:
  - Value 6 indicates Object Communication Model
  - Value 7 indicates Object Access Model

**External Entity in Model.SS_ID**
- Attribute Type: Referential Attribute
- Refers To: Subsystem.SS_ID (R7) (See Page 21)

5. **External Entity Data Item (S_EEDI)**

External Entity Data Item (EEdi_ID, EE_ID, Name, Descrip, DT_ID)

Identifier *: EE_ID, EEdi_ID

Description: Interactions between Objects and External Entities shown in the Object Access Models involve the access of data. An External Entity Data Item is a characteristic of an External Entity that an Object may read.
**External Entity Data Item.EEdi_ID**

Full Name: External Entity Data Item Identifier  
Attribute Type: Base Attribute  
Data Domain: `unique_id`

**External Entity Data Item.EE_ID**

Attribute Type: Referential Attribute  
Refers To: External Entity.EE_ID (R11)  
(See Page 23)

**External Entity Data Item.Name**

Full Name: External Entity Data Item Name  
Attribute Type: Base Attribute  
Data Domain: `string`

**External Entity Data Item.Descrip**

Full Name: External Entity Data Item Description  
Attribute Type: Base Attribute  
Data Domain: `string`

**External Entity Data Item.DT_ID**

Attribute Type: Referential Attribute  
Refers To: Data Type.DT_ID (R15)  
(See Page 29)

---

6. **External Entity Event (S_EEEVT)**

External Entity Event (EEevt_ID, EE_ID, Numb, Mning, Are_KL_C,  
Cust_KL, Drv_Lbl, Descrip)  
Identifier *: EE_ID, EEevt_ID  
Description: An External Entity Event identifies an interaction between an  
Object and an External Entity and is captured on an Object  
Communication Model. Each External Entity Event is given a unique  
label.

**External Entity Event.EEevt_ID**

Full Name: External Entity Entity Event Identifier

---

**CONFIDENTIAL**
Attribute Type: Base Attribute
Data Domain: unique_id

**External Entity Event.EE_ID**
- Attribute Type: Referential Attribute
- Refers To: External Entity.EE_ID (R10)  (See Page 23)

**External Entity Event.Numb**
- Full Name: External Entity Entity Event Number
- Attribute Type: Base Attribute
- Data Domain: integer

**External Entity Event.Mning**
- Full Name: External Entity Entity Event Meaning
- Attribute Type: Base Attribute
- Data Domain: string

**External Entity Event.Are_KL_C**
- Full Name: Are Key Letters Custom Flag
- Attribute Type: Base Attribute
- Data Domain: boolean
- Description: This is a flag that indicates whether custom label keyletters are used for the External Entity Event.
  - Value 0 indicates custom label keyletters are used.
  - Value 1 indicates External Entity keyletters are used.

**External Entity Event.Cust_KL**
- Full Name: Custom External Entity Event Label Key Letters
- Attribute Type: Base Attribute
- Data Domain: string

**External Entity Event.Drv_Lbl**
- Full Name: Derived External Entity Event Label
- Attribute Type: Derived Base Attribute
Data Domain: `string`
Description: Holds the event label - derived by concatenating the keyletters and the event number.

- If the `Are_KL_C` attribute is 0, then the value of the `External Entity.Name` attribute is concatenated with the `External Entity.Numb` attribute.
- If the `Are_KL_C` attribute is 1, then the value of the `External Entity.Cust_KL` attribute is concatenated with the `External Entity.Numb` attribute.

**External Entity Event.Descrip**
Full Name: External Entity Event Description
Attribute Type: Base Attribute
Data Domain: `string`

7. **External Entity Event Data Item (S_EEEDI)**
External Entity Event Data Item (EEedi_ID, EE_ID, Name, Descrip, DT_ID)
Identifier *: EEedi_ID, EE_ID
Description: Synchronous interactions from Objects to External Entities modeled by allowing an Object to synchronously access the data items of the External Entity - the interaction is captured on the Object Communication Model. An External Entity Data Item is a characteristic of an External Entity.

**External Entity Event Data Item.EEedi_ID**
Full Name: External Entity Event Data Item Identifier
Attribute Type: Base Attribute
Data Domain: `unique_id`

**External Entity Event Data Item.EE_ID**
Attribute Type: Referential Attribute
Refers To: External Entity.EE_ID (R12) (See Page 23)
External Entity Event Data Item.Name
Full Name: External Entity Event Data Item Name
Attribute Type: Base Attribute
Data Domain: string

External Entity Event Data Item.Descrip
Full Name: External Entity Event Data Item Description
Attribute Type: Base Attribute
Data Domain: string

External Entity Event Data Item.DT_ID
Attribute Type: Referential Attribute
Refers To: Data Type.DT_ID (R16)  (See Page 29)

8. External Entity Event Data (S_EEEDT)
External Entity Event Data (EE_ID, EEevt_ID, EEedi_ID)
Identifier*: EE_ID, EEevt_ID, EEedi_ID
Description: This object serves as a correlation table.

External Entity Event Data.EE_ID
Attribute Type: Referential Attribute
Refers To: External Entity Event.EE_ID (R13)  (See Page 26)
Refers To: External Entity Event Data Item.EE_ID (R13)  (See Page 27)

External Entity Event Data.EEevt_ID
Attribute Type: Referential Attribute
Refers To: External Entity Event.EEevt_ID (R13)  (See Page 25)

External Entity Event Data.EEedi_ID
Attribute Type: Referential Attribute
Refers To: External Entity Event Data Item.EEedi_ID (R13)  (See Page 27)
9. Data Type (S_DT)

Data Type (DT_ID, Dom_ID, Name, Descrip)
Identifier *: DT_ID
Identifier *2: DT_ID, Dom_ID

Description: An analyst can assign a data type to the various data items in the OOA, e.g., object attribute, state model event data item, transformer/bridge parameter/return value.

This data type does not capture the representation of the data items, but rather, the characteristics of the data items including:
1. Value Definition, e.g., whole numbers
2. Value Range, e.g., values between 0 and 10
3. Operations, e.g., +, -, *, /

**Data Type.DT_ID**

Full Name: Data Type Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

**Data Type.Dom_ID**

Attribute Type: Referential Attribute
Refers To: Domain.Dom_ID (R14) (See Page 20)

**Data Type.Name**

Full Name: Data Type Name
Attribute Type: Base Attribute
Data Domain: string

**Data Type.Descrip**

Full Name: Data Type Description
Attribute Type: Base Attribute
Data Domain: string
10. Core Data Type (S_CDT)

Core Data Type (DT_ID, Core_Typ)
Identifier *: DT_ID
Description: Core Data Types are those data types which are fundamental, or core, to all data types.

Core Data Type.DT_ID
Attribute Type: Referential Attribute
Refers To: Data Type.DT_ID (R17) (See Page 29)

Core Data Type.Core_Typ
Full Name: Core Data Type Core Type
Attribute Type: Base Attribute
Data Domain: integer
Description: Valid Core Types:
   0 = void
   1 = boolean
   2 = integer
   3 = real
   4 = string
   5 = unique_id
   6 = current_state
   7 = same_as_base
   8 = inst_ref<Object>
   9 = inst_ref_set<Object>
  10 = inst<Event>
  11 = inst<Mapping>
  12 = inst_ref<Mapping>

11. User Data Type (S_UDT)

User Data Type (DT_ID, CDT_ID, User_Typ)
Identifier *: DT_ID
Description: User Data Types are those data types which have been derived from the core data types - they typically are derived because more assumptions can be made about the range of values which can be stored or they are derived to serve as a common funnelling point for several data items which share some common data type.

**User Data Type.DT_ID**
- Attribute Type: Referential Attribute
- Refers To: Data Type.DT_ID (R17) (See Page 29)

**User Data Type.CDT_ID**
- Attribute Type: Referential Attribute
- Refers To: Core Data Type.DT_ID (R18) (See Page 30)

**User Data Type.User_Typ**
- Full Name: User Data Type User Type
- Attribute Type: Base Attribute
- Data Domain: `integer`
  - 0 = `user defined`
  - 1 = date
  - 2 = timestamp
  - 3 = inst_ref<Timer>

12. **Bridge (S_BRG)**

Bridge (Brg_ID, EE_ID, Name, Descrip, Brg_Typ, DT_ID)

- Identifier *: Brg_ID

Description: A Bridge is a method associated with an External Entity - bridges can be synchronously called from Action Specifications.

**Bridge.Brg_ID**
- Full Name: Bridge Identifier
- Attribute Type: Base Attribute
- Data Domain: `unique_id`
Step 8: OOA of OOA

**Bridge.EE_ID**
- Attribute Type: Referential Attribute
- Refers To: External Entity.EE_ID (R19)  (See Page 23)

**Bridge.Name**
- Full Name: Bridge Name
- Attribute Type: Base Attribute
- Data Domain: string

**Bridge.Descrip**
- Full Name: Bridge Description
- Attribute Type: Base Attribute
- Data Domain: string

**Bridge.Brg_Typ**
- Full Name: Bridge Type
- Attribute Type: Base Attribute
- Data Domain: integer
  - 0 = user defined
  - 1 = predefined bridge

**Bridge.DT_ID**
- Attribute Type: Referential Attribute
- Refers To: Data Type.DT_ID (R20)  (See Page 29)

13. **Bridge Parameter (S_BPARM)**
- Bridge Parameter (BParm_ID, Brg_ID, Name, DT_ID)
- Identifier *: BParm_ID
- Description: A parameter to a bridge.

**Bridge Parameter.BParm_ID**
- Full Name: Bridge Parameter Identifier
- Attribute Type: Base Attribute
Data Domain: unique_id

**Bridge Parameter.Brg_ID**
- Attribute Type: Referential Attribute
- Refers To: Bridge.Brg_ID (R21) (See Page 31)

**Bridge Parameter.Name**
- Full Name: Bridge Parameter Name
- Attribute Type: Base Attribute
- Data Domain: string

**Bridge Parameter.DT_ID**
- Attribute Type: Referential Attribute
- Refers To: Data Type.DT_ID (R22) (See Page 29)
8.2.1.2 Relationship Descriptions

R1
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Domain is partitioned into Subsystem
Subsystem is first level of partitioning for Domain
Formalized By: Domain.Dom_ID

R2
Relationship Type: Simple
Multiplicity/Conditionality: 1:Mc
Object is contained in Subsystem
Subsystem is decomposed into Object
Formalized By: Subsystem.SS_ID

R3
Relationship Type: Simple
Multiplicity/Conditionality: 1:Mc
Imported Object represents an object from another subsystem in Subsystem
Subsystem can contain objects from other subsystems via Imported Object
Formalized By: Subsystem.SS_ID

R4
Relationship Type: Simple
Multiplicity/Conditionality: 1:Mc
Relationship abstracts associations between objects in Subsystem
Subsystem contains Relationship
Formalized By: Subsystem.SS_ID
### R5

**Relationship Type:** Simple  
**Multiplicity/Conditionality:** 1:Mc  
Communication Path abstracts asynchronous communication between objects in Subsystem  
Subsystem contains Communication Path  
Formalized By: Subsystem.SS_ID

### R6

**Relationship Type:** Simple  
**Multiplicity/Conditionality:** 1:Mc  
Access Path abstracts synchronous data access between objects in Subsystem  
Subsystem contains Access Path  
Formalized By: Subsystem.SS_ID

### R7

**Relationship Type:** Simple  
**Multiplicity/Conditionality:** Mc:1  
Subsystem contains External Entity in Model  
External Entity in Model is a presence of an external entity in Subsystem  
Formalized By: Subsystem.SS_ID

### R8

**Relationship Type:** Simple  
**Multiplicity/Conditionality:** Mc:1  
Domain interacts with External Entity  
External Entity interacts with Domain  
Formalized By: Domain.Dom_ID

### R9

**Relationship Type:** Simple
Step 8: OOA of OOA

Multiplicity/Conditionality: Mc:1
External Entity is represented by External Entity in Model
External Entity in Model is a presence in subsystem model of External Entity
Formalized By: External Entity.EE_ID

R10
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
External Entity can receive asynchronous communication via External Entity Event
External Entity Event is vehicle of communication for External Entity
Formalized By: External Entity.EE_ID

R11
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
External Entity can be accessed synchronously via External Entity Data Item
External Entity Data Item is data for External Entity
Formalized By: External Entity.EE_ID

R12
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
External Entity can asynchronously communicate via External Entity Event Data Item
External Entity Event Data Item is data for events of External Entity
Formalized By: External Entity.EE_ID

R13
Relationship Type: Associative
Multiplicity/Conditionality: 1-(Mc:Mc)
External Entity Event Data Item is carried via External Entity Event
External Entity Event may carry External Entity Event Data Item
Formalized By: External Entity Event.EE_ID, External Entity
   Event.EEevt_ID, External Entity Event Data Item.EEedi_ID, External
   Entity Event Data Item.EE_ID

R14
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Domain contains defined Data Type
Data Type defines types within Domain
Formalized By: Domain.Dom_ID

R15
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of External Entity Data Item
External Entity Data Item is defined by Data Type
Formalized By: Data Type.DT_ID

R16
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of External Entity Event Data Item
External Entity Event Data Item is defined by Data Type
Formalized By: Data Type.DT_ID

R17
Relationship Type: Subtype/Supertype
Subtypes: User Data Type, Core Data Type
Formalized By: Data Type.DT_ID
R18
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Core Data Type defines domain of User Data Type
User Data Type are defined within Core Data Type
Formalized By: Core Data Type.DT_ID

R19
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
External Entity uses Bridge
Bridge provides access to External Entity
Formalized By: External Entity.EE_ID

R20
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the return value Bridge
Bridge return value defined by Data Type
Formalized By: Data Type.DT_ID

R21
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Bridge is part of Bridge Parameter
Bridge Parameter contains Bridge
Formalized By: Bridge.Brg_ID

R22
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of Bridge Parameter
Bridge Parameter is defined by Data Type
Formalized By: Data Type.DT_ID
8.2.2 Subsystem ‘Object’

An Object is the abstraction of real world things that have the same characteristics and conform to a given set of rules. An Object is assigned to exactly one Subsystem. Objects fall into many categories, some of which are tangible things, roles, interactions and specifications. Objects that have interesting behavior are given a lifecycle which is modeled by a State Model.
Step 8: OOA of OOA

112. Transformer (O_TFR)
- Tfr_ID (R115)
- Name
- Descrip
- DT_ID (R116)

113. Transformer Parameter (O_TPARM)
- TParm_ID
- Tfr_ID (R117)
- Name
- DT_ID (R118)

101. Object
- Obj_ID
- Name
- Num
- Key_Lst
- Descrp
- SS_ID (R114)

103. Attribute (O_ATTR)
- Attr_ID
- Obj_ID (R102)
- PAttr_ID (R103)
- Name
- Descrip
- Prefix
- Root_Nm
- Pfx_Mod
- DT_ID (R114)

105. Object Identifier
- Attr_ID (R105)
- Obj_ID (R105)
- Oid_ID (R113)

109. Referential Attribute (O_RATTR)
- Attr_ID (R106)
- Obj_ID (R106)
- RAttr_ID (R113)
- RObj_ID (R113)
- Ref_Mode

110. Attribute Reference in Object (O_REF)
- Obj_ID (R108)
- RObj_ID (R111)
- RAttr_ID (R111)
- ARef_ID
- P_ARef_ID (R112)
- Is_Cstrd
- Descrip

106. Base Attribute (O_BATTR)
- Attr_ID (R106)
- Obj_ID (R108)

107. Derived Base Attribute (O_DBATTR)
- Attr_ID (R107)
- Obj_ID (R107)

108. New Base Attribute (O_NBATTR)
- Attr_ID (R107)
- Obj_ID (R107)

104. Data Type (S_DT)
- DT_ID
- Dom_ID (R14)
- Name
- Descrip

102. Abstract
- Char
- Descrip
- SS_ID (R114)

108. Data Type
- DT_ID
- Name
- Descrip
8.2.2.1 Object and Attribute Descriptions

101. Object (O_OBJ)

Object (Obj_ID, Name, Numb, Key_Lett, Descrip, SS_ID)

Identifier *: Obj_ID

Description: An Object represents an abstraction of a real world thing. All instances of an Object have the same characteristics and conform to the same set of rules. The characteristics of an Object are captured as attributes. Each Object with a Domain are assigned a unique names, numbers, and keyletters.

Object.Obj_ID

Full Name: Object Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Object.Name

Full Name: Object Name
Attribute Type: Base Attribute
Data Domain: string

Object.Numb

Full Name: Object Number
Attribute Type: Base Attribute
Data Domain: integer

Object.Key_Lett

Full Name: Object Keyletters
Attribute Type: Base Attribute
Data Domain: string

Object.Descrip

Full Name: Object Description
Attribute Type: Base Attribute
Data Domain: **string**

**Object.SS_ID**

- Attribute Type: Referential Attribute
- Refers To: Subsystem.SS_ID (R2) (See Page 21)

102. **Imported Object (O_IOBJ)**

- Imported Object (IObj_ID, Obj_ID, Modl_Typ, SS_ID)
- Identifier *: IObj_ID

Description: Objects can have interactions with Objects in other Subsystems. In order to capture these spanning interactions, Objects can be imported into another subsystem. Spanning interactions can be relationships, event communications, or data accesses and are captured in the Object Information Model, Object Communication Model, and Object Access Model, respectively. Spanning interactions provide the data for derivation of the Subsystem Relationship Model, Subsystem Communication Model, and Subsystem Access Model.

**Imported Object.IObj_ID**

- Full Name: Imported Object Identifier
- Attribute Type: Base Attribute
- Data Domain: **unique_id**

**Imported Object.Obj_ID**

- Attribute Type: Referential Attribute
- Refers To: Object.Obj_ID (R101) (See Page 44)

**Imported Object.Modl_Typ**

- Full Name: Model Type
- Attribute Type: Base Attribute
- Data Domain: **integer**

Description: Value indicates what type of model the Imported Object is in:
- Value 5 indicates Object Information Model
- Value 6 indicates Object Communication Model
Value 7 indicates Object Access Model

**Imported Object.SS_ID**

Attribute Type: Referential Attribute

Refers To: Subsystem.SS_ID (R3) (See Page 21)

103. **Attribute (O_ATTR)**

Attribute (Attr_ID, Obj_ID, PAttr_ID, Name, Descrip, Prefix, Root_Nam, Pfx_Mode, DT_ID)

Identifier *: Attr_ID, Obj_ID

Description: An Attribute is an abstraction of a single characteristic possessed by an Object. Usually Objects contain a set of attributes to completely capture all pertinent information. Each Attribute is given a unique name within an Object.

**Attribute.Attr_ID**

Full Name: Attribute Identifier

Attribute Type: Base Attribute

Data Domain: unique_id

**Attribute.Obj_ID**

Attribute Type: Referential Attribute

Refers To: Object.Obj_ID (R102) (See Page 44)

Refers To: Attribute.Obj_ID (R103) (See Page 46)

**Attribute.PAttr_ID**

Attribute Type: Referential Attribute

Refers To: Attribute.Attr_ID (R103) (See Page 46)

**Attribute.Name**

Full Name: Attribute Name

Attribute Type: Base Attribute

Data Domain: string
Attribute.Descrp
   Full Name: Attribute Description
   Attribute Type: Base Attribute
   Data Domain: \textit{string}

Attribute.Prefix
   Full Name: Attribute Name Prefix
   Attribute Type: Base Attribute
   Data Domain: \textit{string}

Attribute.Root_Nam
   Full Name: Attribute Name Root
   Attribute Type: Base Attribute
   Data Domain: \textit{string}

Attribute.Pfx_Mode
   Full Name: Attribute Name Prefix Mode
   Attribute Type: Base Attribute
   Data Domain: \textit{integer}
      \begin{itemize}
         \item 0 = uses no prefix
         \item 1 = uses local prefix
         \item 2 = uses referred to prefix
      \end{itemize}

Attribute.DT_ID
   Attribute Type: Referential Attribute
   Refers To: Data Type.DT_ID (R114) (See Page 29)

104. Object Identifier (O_ID)
   Object Identifier (Oid_ID, Obj_ID)
   Identifier *: Oid_ID, Obj_ID
   Description: A set of one or more Attributes which uniquely distinguishes each instance of an object is an Object Identifier. An Object may have several Identifiers.
**Object Identifier.Oid_ID**

Full Name: Object Identifier Identifier

Attribute Type: Base Attribute

Data Domain: integer

**Object Identifier.Obj_ID**

Attribute Type: Referential Attribute

Refers To: Object.Obj_ID (R104) (See Page 44)

105. **Object Identifier Attribute (O_OIDA)**

Object Identifier Attribute (Attr_ID, Obj_ID, Oid_ID)

Identifier #: Attr_ID, Obj_ID, Oid_ID

Description: An Attribute that is part of an Object Identifier is an Object Identifier Attribute.

**Object Identifier Attribute.Attr_ID**

Attribute Type: Referential Attribute

Refers To: Attribute.Attr_ID (R105) (See Page 46)

**Object Identifier Attribute.Obj_ID**

Attribute Type: Referential Attribute

Refers To: Attribute.Obj_ID (R105) (See Page 46)

Refers To: Object Identifier.Obj_ID (R105) (See Page 48)

**Object Identifier Attribute.Oid_ID**

Attribute Type: Referential Attribute

Refers To: Object Identifier.Oid_ID (R105) (See Page 48)

106. **Base Attribute (O_BATTR)**

Base Attribute (Attr_ID, Obj_ID)

Identifier #: Attr_ID, Obj_ID

Description: A Base Attribute is a non-referential attribute.
Base Attribute.Attr_ID
Attribute Type: Referential Attribute
Refers To: Attribute.Attr_ID (R106)  (See Page 46)

Base Attribute.Obj_ID
Attribute Type: Referential Attribute
Refers To: Attribute.Obj_ID (R106)  (See Page 46)

107. Derived Base Attribute (O_DBATTR)
Derived Base Attribute (Attr_ID, Obj_ID)
Identifier *: Attr_ID, Obj_ID
Description: A Derived Attribute is the result of an algorithm used to derive the value.

Derived Base Attribute.Attr_ID
Attribute Type: Referential Attribute
Refers To: Base Attribute.Attr_ID (R107)  (See Page 46)

Derived Base Attribute.Obj_ID
Attribute Type: Referential Attribute
Refers To: Base Attribute.Obj_ID (R107)  (See Page 46)

108. New Base Attribute (O_NBATTR)
New Base Attribute (Attr_ID, Obj_ID)
Identifier *: Attr_ID, Obj_ID
Description: A New Base Attribute is a non-derived base attribute.

New Base Attribute.Attr_ID
Attribute Type: Referential Attribute
Refers To: Base Attribute.Attr_ID (R107)  (See Page 46)
New Base Attribute.Obj_ID
Attribute Type: Referential Attribute
Refers To: Base Attribute.Obj_ID (R107) (See Page 46)

109. Referential Attribute (O_RATTR)
Referential Attribute (Attr_ID, Obj_ID, BAttr_ID, BObj_ID, Ref_Mode)
Identifier *: Attr_ID, Obj_ID
Description: A Referential Attribute captures the formalization of a relationship. A Referential Attribute refers to an Identifying Attribute in the Object at the other end of the relationship which it formalizes.

Referential Attribute.Attr_ID
Attribute Type: Referential Attribute
Refers To: Attribute.Attr_ID (R106) (See Page 46)

Referential Attribute.Obj_ID
Attribute Type: Referential Attribute
Refers To: Attribute.Obj_ID (R106) (See Page 46)

Referential Attribute.BAttr_ID
Attribute Type: Referential Attribute
Refers To: Base Attribute.Attr_ID (R113) (See Page 49)
Reference IS CONSTRAINED such that Base Attribute related across R113 is same Base Attribute which is found by navigating back through the referred to attributes until the Base Attribute is found.

Referential Attribute.BObj_ID
Attribute Type: Referential Attribute
Refers To: Base Attribute.Obj_ID (R113) (See Page 49)
Reference IS CONSTRAINED such that Base Attribute related across R113 is same Base Attribute which is found by navigating back through the referred to attributes until the Base Attribute is found.
Referential Attribute.Ref_Mode

Full Name: Referential Attribute Mode
Attribute Type: Base Attribute
Data Domain: `integer`

110. Attribute Reference in Object (O_REF)

Attribute Reference in Object (Obj_ID, RObj_ID, ROid_ID, RAttr_ID, Rel_ID, OIR_ID, ROIR_ID, Attr_ID, ARef_ID, PARRef_ID, Is_Cstrd, Descrip)
Identifier *: Obj_ID, RObj_ID, ROid_ID, RAttr_ID, Rel_ID, OIR_ID, ROIR_ID
Identifier *2: ARef_ID
Description: The Object represents an R# which follows a referential attribute.

Attribute Reference in Object.Obj_ID
Attribute Type: Referential Attribute
Refers To: Referential Attribute.Obj_ID (R108) (See Page 50)
Refers To: Referring Object in Rel.Obj_ID (R111) (See Page 66)

Attribute Reference in Object.RObj_ID
Attribute Type: Referential Attribute
Refers To: Referred To Identifier Attribute.Obj_ID (R111) (See Page 53)

Attribute Reference in Object.ROid_ID
Attribute Type: Referential Attribute
Refers To: Referred To Identifier Attribute.Oid_ID (R111) (See Page 53)

Attribute Reference in Object.RAttr_ID
Attribute Type: Referential Attribute
Refers To: Referred To Identifier Attribute.Attr_ID (R111) (See Page 53)

Attribute Reference in Object.Rel_ID
Attribute Type: Referential Attribute
Refer To: Referring Object in Rel.Rel_ID (R111) (See Page 66)
Refer To: Referred To Identifier Attribute.Rel_ID (R111) (See Page 53)

**Attribute Reference in Object.OIR_ID**
Attribute Type: Referential Attribute
Refer To: Referring Object in Rel.OIR_ID (R111) (See Page 66)

**Attribute Reference in Object.ROIR_ID**
Attribute Type: Referential Attribute
Refer To: Referred To Identifier Attribute.OIR_ID (R111) (See Page 53)

**Attribute Reference in Object.Attr_ID**
Attribute Type: Referential Attribute
Refer To: Referential Attribute.Attr_ID (R108) (See Page 50)

**Attribute Reference in Object.ARef_ID**
Full Name: Object Identifier Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

**Attribute Reference in Object.PARef_ID**
Attribute Type: Referential Attribute
Refer To: Attribute Reference in Object.ARef_ID (R112) (See Page 52)

**Attribute Reference in Object.Is_Cstrd**
Full Name: Attribute Reference in Object Is Constrained Flag
Attribute Type: Base Attribute
Data Domain: boolean
  0 = not constrained
  1 = constrained

**Attribute Reference in Object.Descrip**
Full Name: Attribute Reference in Object Description
Attribute Type: Base Attribute
111. Referred To Identifier Attribute (O_RTIDA)

Referred To Identifier Attribute (Attr_ID, Obj_ID, Oid_ID, Rel_ID, OIR_ID)
Identifier #: Obj_ID, Attr_ID, Oid_ID, Rel_ID, OIR_ID
Description: This object serves a linkage between the R# (Attribute Reference in Object) and the referred to Object Identifier Attribute.

Referred To Identifier Attribute.Attr_ID
Attribute Type: Referential Attribute
Refers To: Object Identifier Attribute.Attr_ID (R110) (See Page 48)

Referred To Identifier Attribute.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object Identifier Attribute.Obj_ID (R110) (See Page 48)
Refers To: Referred To Object in Rel.Obj_ID (R110) (See Page 65)

Referred To Identifier Attribute.Oid_ID
Attribute Type: Referential Attribute
Refers To: Object Identifier Attribute.Oid_ID (R110) (See Page 48)
Refers To: Referred To Object in Rel.Obj_ID (R110) (See Page 66)

Referred To Identifier Attribute.Rel_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Rel_ID (R110) (See Page 65)

Referred To Identifier Attribute.OIR_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.OIR_ID (R110) (See Page 66)

112. Transformer (O_TFR)
Transformer (Tfr_ID, Obj_ID, Name, Descrip, DT_ID)
Identifier *: Tfr_ID
Description: A Transformer is a method associated with an Object - transformers can be synchronously called from Action Specifications.

Transformer.Tfr_ID
Full Name: Transformer Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Transformer.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object.Obj_ID (R115)  (See Page 44)

Transformer.Name
Full Name: Transformer Name
Attribute Type: Base Attribute
Data Domain: string

Transformer.Descrip
Full Name: Transformer Description
Attribute Type: Base Attribute
Data Domain: string

Transformer.DT_ID
Attribute Type: Referential Attribute
Refers To: Data Type.DT_ID (R116)  (See Page 29)

113. Transformer Parameter (O_TPARM)
Transformer Parameter (TParm_ID, Tfr_ID, Name, DT_ID)
Identifier *: TParm_ID
Description: This object is a parameter to a transformer.
**Transformer Parameter.TParm_ID**

- Full Name: Transformer Parameter Identifier
- Attribute Type: Base Attribute
- Data Domain: `unique_id`

**Transformer Parameter.Tfr_ID**

- Attribute Type: Referential Attribute
- Refers To: Transformer.Tfr_ID (R117) (See Page 54)

**Transformer Parameter.Name**

- Full Name: Transformer Parameter Name
- Attribute Type: Base Attribute
- Data Domain: `string`

**Transformer Parameter.DT_ID**

- Attribute Type: Referential Attribute
- Refers To: Data Type.DT_ID (R118) (See Page 29)
8.2.2.2 Relationship Descriptions

**R101**
- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- Object has presence in other subsystems via Imported Object
- Imported Object represents Object
- Formalized By: Object.Obj_ID

**R102**
- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- Object has characteristics abstracted by Attribute
- Attribute abstracts characteristics of Object
- Formalized By: Object.Obj_ID

**R103**
- Relationship Type: Simple
- Multiplicity/Conditionality: 1c:1c
- Attribute precedes Attribute
- Attribute succeeds Attribute
- Formalized By: Attribute.Attr_ID, Attribute.Obj_ID

**R104**
- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- Object is identified by Object Identifier
- Object Identifier identifies Object
- Formalized By: Object.Obj_ID
R105
Relationship Type: Associative
Multiplicity/Conditionality: 1-(Mc:Mc)
Object Identifier is made up of Attribute
Attribute is part of Object Identifier

R106
Relationship Type: Subtype/Supertype
Subtypes: Base Attribute, Referential Attribute
Formalized By: Attribute.Attr_ID, Attribute.Obj_ID

R107
Relationship Type: Subtype/Supertype
Subtypes: Derived Base Attribute, New Base Attribute
Formalized By: Base Attribute.Attr_ID, Base Attribute.Obj_ID

R108
Relationship Type: Simple
Multiplicity/Conditionality: M:1
Referential Attribute resolves Attribute Reference in Object
Attribute Reference in Object is resolved by Referential Attribute
Formalized By: Referential Attribute.Attr_ID, Referential Attribute.Obj_ID

R109
Relationship Type: Simple
Multiplicity/Conditionality: 1c:Mc
Referred To Object in Rel is identified in this relationship by Object Identifier
Object Identifier identifies for this relationship Referred To Object in Rel
Formalized By: Object Identifier.Oid_ID, Object Identifier.Obj_ID
Step 8: OOA of OOA

R110
Relationship Type: Associative
Multiplicity/Conditionality: 1-(M:Mc)
Referred To Object in Rel is identified in this relationship by Object Identifier Attribute
Object Identifier Attribute identifies for this relationship Referred To Object in Rel
Formalized By: Referred To Object in Rel.Obj_ID, Referred To Object in Rel.Rel_ID, Referred To Object in Rel.OIR_ID, Referred To Object in Rel.Oid_ID, Object Identifier Attribute.Attr_ID, Object Identifier Attribute.Obj_ID, Object Identifier Attribute.Oid_ID

R111
Relationship Type: Associative
Multiplicity/Conditionality: 1-(M:M)
Referring Object in Rel refers across relationship via Referred To Identifier Attribute
Referred To Identifier Attribute is referred to object by Referring Object in Rel
Formalized By: Referred To Identifier Attribute.Obj_ID, Referred To Identifier Attribute.Attr_ID, Referred To Identifier Attribute.Oid_ID, Referred To Identifier Attribute.Rel_ID, Referred To Identifier Attribute.OIR_ID, Referring Object in Rel Obj_ID, Referring Object in Rel Rel_ID, Referring Object in Rel OIR_ID

R112
Relationship Type: Simple
Multiplicity/Conditionality: 1c:1c
Attribute Reference in Object succeeds Attribute Reference in Object
Attribute Reference in Object precedes Attribute Reference in Object
Formalized By: Attribute Reference in Object.ARef_ID

R113
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Base Attribute can be the base of Referential Attribute
Referential Attribute navigates back to Base Attribute
Formalized By: Base Attribute.Attr_ID, Base Attribute.Obj_ID

R114
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of Attribute
Attribute is defined by Data Type
Formalized By: Data Type.DT_ID

R115
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Object may contain Transformer
Transformer is associated with Object
Formalized By: Object.Obj_ID

R116
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of the return code Transformer
Transformer return code is defined by Data Type
Formalized By: Data Type.DT_ID

R117
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Transformer contains Transformer Parameter
Transformer Parameter is part of a Transformer
Formalized By: Transformer.Tfr_ID

R118
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of Transformer Parameter
Transformer Parameter is defined by Data Type
Formalized By: Data Type.DT_ID
8.2.3 Subsystem ‘Relationship’
A Relationship captures an association between things in the real world. A Relationship is stated in terms of the formal objects that model the real world entities participating in the association. There can be any number of Relationships between the same two objects and any object can participate in any number of Relationships with other objects.
201. Relationship (R_REL)
- Rel_ID
- Name
- Desc
- SS_ID (R4)

205. Simple Relationship (R_SWP)
- Rel_ID (R206)

206. Association Relationship (R_ASSOC)
- Rel_ID (R206)

208. Subtype/Supertype Relationship (R_SUBSUP)
- Rel_ID (R206)

212. Composition Relationship (R_COMP)
- Rel_ID (R206)
- Rel_Ch
8.2.3.1 Object and Attribute Descriptions

201. Relationship (R_REL)

Relationship (Rel_ID, Numb, Descrip, SS_ID)
Identifier *: Rel_ID
Description: A Relationship captures an association that exists between things in the real world. A Relationship is stated in terms of the formal Objects that participate in the association.

- **Relationship.Rel_ID**
  - Full Name: Relationship Identifier
  - Attribute Type: Base Attribute
  - Data Domain: unique_id

- **Relationship.Numb**
  - Full Name: Relationship Number
  - Attribute Type: Base Attribute
  - Data Domain: integer

- **Relationship.Descip**
  - Full Name: Relationship Description
  - Attribute Type: Base Attribute
  - Data Domain: string

- **Relationship.SS_ID**
  - Attribute Type: Referential Attribute
  - Refers To: Subsystem.SS_ID (R4)  (See Page 21)

202. Object in Relationship (R_OIR)

Object in Relationship (Obj_ID, Rel_ID, OIR_ID, IObj_ID)
Identifier *: Obj_ID, Rel_ID, OIR_ID
Description: An Object in Relationship captures the role which an object plays in participating in a relationship.
Object in Relationship.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object.Obj_ID (R201) (See Page 44)

Object in Relationship.Rel_ID
Attribute Type: Referential Attribute
Refers To: Relationship.Rel_ID (R201) (See Page 64)

Object in Relationship.OIR_ID
Full Name: Object in Relationship Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

AObject in Relationship.IObj_ID
Attribute Type: Referential Attribute
Refers To: Imported Object. IObj_ID (R202) (See Page 45)

203. Referred To Object in Rel (R_RTO)
Referred To Object in Rel (Obj_ID, Rel_ID, OIR_ID, Oid_ID)
Identifier *: Obj_ID, Rel_ID, OIR_ID
Identifier *2: Obj_ID, Rel_ID, OIR_ID, Oid_ID
Description: A Referred To Object in Relationship is an object which contains identifier attributes which are referred to across the relationship.

Referred To Object in Rel.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Obj_ID (R203) (See Page 65)

Referred To Object in Rel.Rel_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Rel_ID (R203) (See Page 65)
Step 8: OOA of OOA

Referred To Object in Rel.OIR_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.OIR_ID (R203)  (See Page 65)

Referred To Object in Rel.Oid_ID
Attribute Type: Referential Attribute
Refers To: Object Identifier.Oid_ID (R109)  (See Page 48)

204. Referring Object in Rel (R_RGO)
Referring Object in Rel (Obj_ID, Rel_ID, OIR_ID)
Identifier *: Obj_ID, Rel_ID, OIR_ID
Description: A referring Object in Relationship is an object which contains referential attributes which refer to identifying attributes across the relationship.

Referring Object in Rel.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Obj_ID (R203)  (See Page 65)

Referring Object in Rel.Rel_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Rel_ID (R203)  (See Page 65)

Referring Object in Rel.OIR_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.OIR_ID (R203)  (See Page 65)

205. Simple Relationship (R_SIMP)
Simple Relationship (Rel_ID)
Identifier *: Rel_ID
Description: A Simple Relationship is a relationship between two objects which is formalized with referential attributes.
Simple Relationship.Rel_ID
Attribute Type: Referential Attribute
Refers To: Relationship.Rel_ID (R206) (See Page 64)

206. Object As Simple Participant (R_PART)
Object As Simple Participant (Obj_ID, Rel_ID, OIR_ID, Mult, Cond, Txt_Phrs)
Identifier *: Rel_ID, Obj_ID, OIR_ID
Description: An Object As Simple Participant is the referred to object in a simple relationship.

Object As Simple Participant.Obj_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Obj_ID (R204) (See Page 65)

Object As Simple Participant.Rel_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Rel_ID (R204) (See Page 65)

Object As Simple Participant.OIR_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.OIR_ID (R204) (See Page 66)

Object As Simple Participant.Mult
Full Name: Multiplicity
Attribute Type: Base Attribute
Data Domain: integer
0 = one
1 = many

Object As Simple Participant.Cond
Full Name: Conditionality
Attribute Type: Base Attribute
Data Domain: **integer**

- 0 = uncond
- 1 = cond

**Object As Simple Participant.Txt_Phrs**

- Full Name: Text Phrase
- Attribute Type: Base Attribute
- Data Domain: **string**

207. **Object As Simple Formalizer (R_FORM)**

Object As Simple Formalizer (Obj_ID, Rel_ID, OIR_ID, Mult, Cond, Txt_Phrs)

- Identifier *: Rel_ID, Obj_ID, OIR_ID
- Description: An Object As Simple Formalizer is the referring object in a simple relationship.

**Object As Simple Formalizer.Obj_ID**

- Attribute Type: Referential Attribute
- Refers To: Referring Object in Rel.Obj_ID (R205) (See Page 66)

**Object As Simple Formalizer.Rel_ID**

- Attribute Type: Referential Attribute
- Refers To: Referring Object in Rel.Rel_ID (R205) (See Page 65)

**Object As Simple Formalizer.OIR_ID**

- Attribute Type: Referential Attribute
- Refers To: Referring Object in Rel.OIR_ID (R205) (See Page 66)

**Object As Simple Formalizer.Mult**

- Full Name: Multiplicity
- Attribute Type: Base Attribute
- Data Domain: **integer**

- 0 = one
1 = many

**Object As Simple Formalizer.Cond**
Full Name: Conditionality
Attribute Type: Base Attribute
Data Domain: integer
  0 = uncond
  1 = cond

**Object As Simple Formalizer.Txt_Phrs**
Full Name: Text Phrase
Attribute Type: Base Attribute
Data Domain: string

208. **Associative Relationship (R_ASSOC)**
  Associative Relationship (Rel_ID)
  Identifier *: Rel_ID

**Associative Relationship.Rel_ID**
  Attribute Type: Referential Attribute
  Refers To: Relationship.Rel_ID (R206)

209. **Object As Associated One Side (R_AONE)**
  Object As Associated One Side (Obj_ID, Rel_ID, OIR_ID, Mult, Cond,
  Txt_Phrs)
  Identifier *: Rel_ID, Obj_ID, OIR_ID

**Object As Associated One Side.Obj_ID**
  Attribute Type: Referential Attribute
  Refers To: Referred To Object in Rel.Obj_ID (R204) (See Page 65)

**Object As Associated One Side.Rel_ID**
  Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Rel_ID (R204)  (See Page 65)

**Object As Associated One Side.OIR_ID**
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.OIR_ID (R204)

**Object As Associated One Side.Mult**
Full Name: Multiplicity
Attribute Type: Base Attribute
Data Domain: integer
   0 = one
   1 = many

**Object As Associated One Side.Cond**
Full Name: Conditionality
Attribute Type: Base Attribute
Data Domain: integer
   0 = unconditional
   1 = conditional

**Object As Associated One Side.Txt_Phrs**
Full Name: Text Phrase
Attribute Type: Base Attribute
Data Domain: string

210. **Object As Associated Other Side (R_AOTH)**
Object As Associated Other Side (Obj_ID, Rel_ID, OIR_ID, Mult, Cond, Txt_Phrs)
Identifier *: Rel_ID, Obj_ID, OIR_ID

**Object As Associated Other Side.Obj_ID**
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Obj_ID (R204)  (See Page 65)
Object As Associated Other Side. Rel_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Rel_ID (R204)  (See Page 65)

Object As Associated Other Side. OIR_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.OIR_ID (R204)  (See Page 66)

Object As Associated Other Side. Mult
Full Name: Multiplicity
Attribute Type: Base Attribute
Data Domain: integer
  0 = one
  1 = many

Object As Associated Other Side. Cond
Full Name: Conditionality
Attribute Type: Base Attribute
Data Domain: integer
  0 = unconditional
  1 = conditional

Object As Associated Other Side. Txt_Phrs
Full Name: Text Phrase
Attribute Type: Base Attribute
Data Domain: string

211. Object As Associator (R_ASSR)
Object As Associator (Obj_ID, Rel_ID, OIR_ID, Mult)
Identifier *: Rel_ID, Obj_ID, OIR_ID

Object As Associator. Obj_ID
Attribute Type: Referential Attribute
Refers To: Referring Object in Rel.Obj_ID (R205)  (See Page 66)

Object As Associator Rel_ID
Attribute Type: Referential Attribute
Refers To: Referring Object in Rel.Rel_ID (R205)  (See Page 66)

Object As Associator OIR_ID
Attribute Type: Referential Attribute
Refers To: Referring Object in Rel.OIR_ID (R205)  (See Page 66)

Object As Associator Mult
Full Name: Multiplicity
Attribute Type: Base Attribute
Data Domain: integer
0 = one
1 = many

212. Subtype/Supertype Relationship (R_SUBSUP)
Subtype/Supertype Relationship Rel_ID
Identifier *: Rel_ID

Subtype/Supertype Relationship Rel_ID
Attribute Type: Referential Attribute
Refers To: Relationship Rel_ID (R206)  (See Page 64)

213. Object As Supertype (R_SUPER)
Object As Supertype Obj_ID, Rel_ID, OIR_ID
Identifier *: Rel_ID, Obj_ID, OIR_ID

Object As Supertype Obj_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Obj_ID (R204)  (See Page 65)
Object As Supertype.Rel_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.Rel_ID (R204)  (See Page 65)

Object As Supertype.OIR_ID
Attribute Type: Referential Attribute
Refers To: Referred To Object in Rel.OIR_ID (R204)  (See Page 66)

214. Object As Subtype (R_SUB)
Object As Subtype (Obj_ID, Rel_ID, OIR_ID)
Identifier *: Rel_ID, Obj_ID, OIR_ID

Object As Subtype.Obj_ID
Attribute Type: Referential Attribute
Refers To: Referring Object in Rel.Obj_ID (R205)  (See Page 66)

Object As Subtype.Rel_ID
Attribute Type: Referential Attribute
Refers To: Referring Object in Rel.Rel_ID (R205)  (See Page 66)

Object As Subtype.OIR_ID
Attribute Type: Referential Attribute
Refers To: Referring Object in Rel.OIR_ID (R205)  (See Page 66)

215. Composition Relationship (R_COMP)
Composition Relationship (Rel_ID, Rel_Chn)
Identifier *: Rel_ID

Composition Relationship.Rel_ID
Attribute Type: Referential Attribute
Refers To: Relationship.Rel_ID (R206)  (See Page 64)
Composition Relationship.Rel_Chn
Full Name: Relationship Chain
Attribute Type: Base Attribute
Data Domain: string

216. Object As Composition One Side (R_CONE)
Object As Composition One Side (Obj_ID, Rel_ID, OIR_ID, Mult, Cond, Txt_Phrs)
Identifier *: Rel_ID, Obj_ID, OIR_ID

Object As Composition One Side.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Obj_ID (R203) (See Page 65)

Object As Composition One Side.Rel_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Rel_ID (R203) (See Page 65)

Object As Composition One Side.OIR_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.OIR_ID (R203) (See Page 65)

Object As Composition One Side.Mult
Full Name: Multiplicity
Attribute Type: Base Attribute
Data Domain: integer
0 = one
1 = many

Object As Composition One Side.Cond
Full Name: Conditionality
Attribute Type: Base Attribute
Data Domain: integer
0 = unconditional
1 = conditional

Object As Composition One Side.Txt_Phrs
Full Name: Text Phrase
Attribute Type: Base Attribute
Data Domain: string

217. Object As Composition Other Side (R_COTH)
Object As Composition Other Side (Obj_ID, Rel_ID, OIR_ID, Mult, Cond, Txt_Phrs)
Identifier *: Rel_ID, Obj_ID, OIR_ID

Object As Composition Other Side.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Obj_ID (R203)  (See Page 65)

Object As Composition Other Side.Rel_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.Rel_ID (R203)  (See Page 65)

Object As Composition Other Side.OIR_ID
Attribute Type: Referential Attribute
Refers To: Object in Relationship.OIR_ID (R203)  (See Page 65)

Object As Composition Other Side.Mult
Full Name: Multiplicity
Attribute Type: Base Attribute
Data Domain: integer
0 = one
1 = many
Object As Composition Other Side.Cond
Full Name: Conditionality
Attribute Type: Base Attribute
Data Domain: integer
0 = unconditional
1 = conditional

Object As Composition Other Side.Txt_Phrs
Full Name: Text Phrase
Attribute Type: Base Attribute
Description: string
8.2.3.2 Relationship Descriptions

R201
Relationship Type: Associative
Multiplicity/Conditionality: M-(M:Mc)
Relationship abstracts association between instances of Object
Object has instance associations abstracted by Relationship
Formalized By: Object.Obj_ID, Relationship.Rel_ID

R202
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1c
Imported Object is used for spanning relationships as Object in Relationship
Object in Relationship may be represented by Imported Object
Formalized By: Imported Object.Obj_ID

R203
Relationship Type: Subtype/Supertype
Subtypes: Referred To Object in Rel, Referring Object in Rel, Object As Composition One Side, Object As Composition Other Side
Formalized By: Object in Relationship.Obj_ID, Object in Relationship.Rel_ID, Object in Relationship.OIR_ID

R204
Relationship Type: Subtype/Supertype
Subtypes: Object As Simple Participant, Object As Associated One Side, Object As Associated Other Side, Object As Supertype
Formalized By: Referred To Object in Rel.Obj_ID, Referred To Object in Rel.Rel_ID, Referred To Object in Rel.OIR_ID
R205
Relationship Type: Subtype/Supertype
Subtypes: Object As Simple Formalizer, Object As Associator, Object As Subtype
Formalized By: Referring Object in Rel.Obj_ID, Referring Object in Rel.Rel_ID, Referring Object in Rel.OIR_ID

R206
Relationship Type: Subtype/Supertype
Subtypes: Simple Relationship, Associative Relationship, Subtype/Supertype Relationship, Composition Relationship
Formalized By: Relationship.Rel_ID

R207
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Simple Relationship relates Object As Simple Participant
Object As Simple Participant is related to formalizer via Simple Relationship
Formalized By: Simple Relationship.Rel_ID

R208
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Simple Relationship relates Object As Simple Formalizer
Object As Simple Formalizer is related to participant via Simple Relationship
Formalized By: Simple Relationship.Rel_ID

R209
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Associative Relationship relates Object As Associated One Side
Object As Associated One Side is related to other side via Associative Relationship
Formalized By: Associative Relationship.Rel_ID

\[ \Leftrightarrow \]
**R210**
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Associative Relationship relates Object As Associated Other Side
Object As Associated Other Side is related to one side via Associative Relationship
Formalized By: Associative Relationship.Rel_ID

\[ \Leftrightarrow \]
**R211**
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Associative Relationship uses a formalizer Object As Associator
Object As Associator formalizes relationship between associated objects
Formalized By: Associative Relationship.Rel_ID

\[ \Leftrightarrow \]
**R212**
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Subtype/Supertype Relationship relates Object As Supertype
Object As Supertype is related to subtypes via Subtype/Supertype Relationship
Formalized By: Subtype/Supertype Relationship.Rel_ID

\[ \Leftrightarrow \]
**R213**
Relationship Type: Simple
Multiplicity/Conditionality: M:1
Subtype/Supertype Relationship relates Object As Subtype
Object As Subtype is related to supertype via Subtype/Supertype Relationship
Formalized By: Subtype/Supertype Relationship.Rel_ID

\[ \text{R214} \]
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Composition Relationship relates Object As Composition One Side
Object As Composition One Side is related to other side via Composition Relationship
Formalized By: Composition Relationship.Rel_ID

\[ \text{R215} \]
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Composition Relationship relates Object As Composition Other Side
Object As Composition Other Side is related to one side via Composition Relationship
Formalized By: Composition Relationship.Rel_ID
8.2.4 Subsystem ‘Communication & Access’

Interactions between Objects is modeled by Communication and Access Paths. Communication Paths model Active Objects that interact via events. Access Paths model Objects that interact by accessing another Object’s attributes. An Access Path must originate from an Active Object. Objects also may interact with entities outside of the Domain being modeled. Such interactions are show with Communication and Access Paths between Objects and External Entities. Communication and Access Paths may also cross Subsystem boundaries.
8.2.4.1 Object and Attribute Descriptions

401. Communication Path (CA_COMM)
Communication Path (CPath_ID, SS_ID)
Identifier *: CPath_ID

Communication Path.CPath_ID
Full Name: Communication Path Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Communication Path.SS_ID
Attribute Type: Referential Attribute
Refers To: Subsystem.SS_ID (R5) (See Page 21)

402. EE to SM Comm Path (CA_EESMC)
EE to SM Comm Path (CPath_ID, EEmod_ID, EE_ID, SM_ID)
Identifier *: CPath_ID
Identifier *2: EE_ID, SM_ID

EE to SM Comm Path.CPath_ID
Attribute Type: Referential Attribute
Refers To: Communication Path.CPath_ID (R401) (See Page 84)

EE to SM Comm Path.EEmod_ID
Attribute Type: Referential Attribute
Refers To: External Entity in Model.EEmod_ID (R402) (See Page 24)

EE to SM Comm Path.EE_ID
Attribute Type: Referential Attribute
Refers To: External Entity in Model.EE_ID (R402) (See Page 24)
EE to SM Comm Path.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R403) (See Page 100)

403. SM to SM Comm Path (CA_SMSMC)
SM to SM Comm Path (CPath_ID, OSM_ID, DSM_ID, OIObj_ID, DIObj_ID)
Identifier *: CPath_ID
Identifier *2: OSM_ID, DSM_ID

SM to SM Comm Path.CPath_ID
Attribute Type: Referential Attribute
Refers To: Communication Path.CPath_ID (R401) (See Page 84)

SM to SM Comm Path.OSM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R406) (See Page 100)

SM to SM Comm Path.DSM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R407) (See Page 100)

SM to SM Comm Path.OIObj_ID
Attribute Type: Referential Attribute
Refers To: Imported Object.IObj_ID (R424) (See Page 45)

SM to SM Comm Path.DIObj_ID
Attribute Type: Referential Attribute
Refers To: Imported Object.IObj_ID (R414) (See Page 45)

404. SM to EE Comm Path (CA_SMEEC)
SM to EE Comm Path (CPath_ID, SM_ID, EE_ID, EEmod_ID)
86  Step 8: OOA of OOA

Identifier *: CPath_ID
Identifier *2: SM_ID, EE_ID

SM to EE Comm Path.CPath_ID
  Attribute Type: Referential Attribute
  Refers To: Communication Path.CPath_ID (R401) (See Page 84)

SM to EE Comm Path.SM_ID
  Attribute Type: Referential Attribute
  Refers To: State Model.SM_ID (R410) (See Page 100)

SM to EE Comm Path.EE_ID
  Attribute Type: Referential Attribute
  Refers To: External Entity in Model.EE_ID (R411) (See Page 24)

SM to EE Comm Path.EEmod_ID
  Attribute Type: Referential Attribute
  Refers To: External Entity in Model.EEmod_ID (R411) (See Page 24)

405. EE to SM Event Comm (CA_EESME)
EE to SM Event Comm (CPath_ID, SM_ID, SMevt_ID)
Identifier *: SM_ID, SMevt_ID, CPath_ID

EE to SM Event Comm.CPath_ID
  Attribute Type: Referential Attribute
  Refers To: EE to SM Comm Path.CPath_ID (R404) (See Page 84)

EE to SM Event Comm.SM_ID
  Attribute Type: Referential Attribute
  Refers To: State Model Event.SM_ID (R405) (See Page 102)

EE to SM Event Comm.SMevt_ID
  Attribute Type: Referential Attribute
406. SM to SM Event Comm (CA_SMSME)
SM to SM Event Comm (CPath_ID, SM_ID, SMevt_ID)
Identifier *: SM_ID, SMevt_ID, CPath_ID

SM to SM Event Comm.CPath_ID
Attribute Type: Referential Attribute
Refers To: SM to SM Comm Path.CPath_ID (R408) (See Page 85)

SM to SM Event Comm.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model Event.SM_ID (R409) (See Page 102)

SM to SM Event Comm.SMevt_ID
Attribute Type: Referential Attribute
Refers To: State Model Event.SMevt_ID (R409) (See Page 102)

407. SM to EE Event Comm (CA_SMEEE)
SM to EE Event Comm (CPath_ID, EE_ID, EEevt_ID)
Identifier *: EE_ID, EEevt_ID, CPath_ID

SM to EE Event Comm.CPath_ID
Attribute Type: Referential Attribute
Refers To: SM to EE Comm Path.CPath_ID (R412) (See Page 86)

SM to EE Event Comm.EE_ID
Attribute Type: Referential Attribute
Refers To: External Entity Event.EE_ID (R413) (See Page 26)

SM to EE Event Comm.EEevt_ID
Attribute Type: Referential Attribute
88

Step 8: OOA of OOA

Refers To: External Entity Event.EEevt_ID (R413)  (See Page 25)

408. **Access Path (CA_ACC)**

Access Path (APath_ID, SS_ID, SM_ID, IObj_ID)
Identifier #: APath_ID

**Access Path.APath_ID**
Full Name: Access Path Identifier
Attribute Type: Base Attribute
Data Domain: \texttt{unique_id}

**Access Path.SS_ID**
Attribute Type: Referential Attribute
Refers To: Subsystem.SS_ID (R6)  (See Page 21)

**Access Path.SM_ID**
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R416)  (See Page 101)

**Access Path.IObj_ID**
Attribute Type: Referential Attribute
Refers To: Imported Object.IObj_ID (R425)  (See Page 45)

409. **SM to OBJ Access Path (CA_SMOA)**

SM to OBJ Access Path (APath_ID, Obj_ID, IObj_ID)
Identifier #: APath_ID, Obj_ID

**SM to OBJ Access Path.APath_ID**
Attribute Type: Referential Attribute
Refers To: Access Path.APath_ID (R415)  (See Page 88)
SM to OBJ Access Path.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object.Obj_ID (R417) (See Page 44)

SM to OBJ Access Path.IObj_ID
Attribute Type: Referential Attribute
Refers To: Imported Object.IObj_ID (R420) (See Page 45)

410. SM to EE Access Path (CA_SMEEA)
SM to EE Access Path (APath_ID, EE_ID, EEmod_ID)
Identifier *: APath_ID, EE_ID

SM to EE Access Path.APath_ID
Attribute Type: Referential Attribute
Refers To: Access Path.APath_ID (R415) (See Page 88)

SM to EE Access Path.EE_ID
Attribute Type: Referential Attribute
Refers To: External Entity in Model.EE_ID (R421) (See Page 24)

SM to EE Access Path.EEmod_ID
Attribute Type: Referential Attribute
Refers To: External Entity in Model.EEmod_ID (R421) (See Page 24)

411. SM to OBJ Attribute Access (CA_SMOAA)
SM to OBJ Attribute Access (APath_ID, Attr_ID, Obj_ID)
Identifier *: Obj_ID, Attr_ID, APath_ID

ASM to OBJ Attribute Access.APath_ID
Attribute Type: Referential Attribute
Refers To: SM to OBJ Access Path.APath_ID (R418) (See Page 88)
**SM to OBJ Attribute Access.Attr_ID**
- Attribute Type: Referential Attribute
- Refers To: Attribute.Attr_ID (R419)  (See Page 46)

**SM to OBJ Attribute Access.Obj_ID**
- Attribute Type: Referential Attribute
- Refers To: SM to OBJ Access Path.Obj_ID (R418)  (See Page 89)
- Refers To: Attribute.Obj_ID (R419)  (See Page 46)
- Refers To: SM to OBJ Access Path.Obj_ID (R418)  (See Page 89)

### 412. SM to EE Data Item Access (CA_SMEED)

- **SM to EE Data Item Access.(APath_ID, EEdi_ID, EE_ID)**
- **Identifier *:** EE_ID, EEdi_ID, APath_ID

**SM to EE Data Item Access.APath_ID**
- Attribute Type: Referential Attribute
- Refers To: SM to EE Access Path.APath_ID (R422)  (See Page 89)

**SM to EE Data Item Access.EEdi_ID**
- Attribute Type: Referential Attribute
- Refers To: External Entity Data Item.EEdi_ID (R423)  (See Page 25)

**SM to EE Data Item Access.EE_ID**
- Attribute Type: Referential Attribute
- Refers To: External Entity Data Item.EE_ID (R423)  (See Page 25)
- Refers To: SM to EE Access Path.EE_ID (R422)  (See Page 89)
8.2.4.2 Relationship Descriptions

**R401**

Relationship Type: Subtype/Supertype
Subtypes: EE to SM Comm Path, SM to SM Comm Path, SM to EE Comm Path
Formalized By: Communication Path.CPath_ID

**R402**

Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
External Entity in Model originates EE to SM Comm Path
EE to SM Comm Path originates from External Entity in Model
Formalized By: External Entity in Model.EE_ID, External Entity in Model.EEmod_ID

**R403**

Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model has received event communication represented by EE to SM Comm Path
EE to SM Comm Path shows event communication to State Model
Formalized By: State Model.SM_ID

**R404**

Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
EE to SM Comm Path carries EE to SM Event Comm
EE to SM Event Comm is carried by EE to SM Comm Path
Formalized By: EE to SM Comm Path.CPath_ID
R405
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model Event is carried to other SMs via EE to SM Event Comm
EE to SM Event Comm represents communication of State Model Event
Formalized By: State Model Event.SM_ID, State Model Event.SMevt_ID

R406
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model originates SM to SM Comm Path
SM to SM Comm Path originates from State Model
Formalized By: State Model.SM_ID

R407
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model has received event communication represented by SM to SM Comm Path
SM to SM Comm Path shows event communication to State Model
Formalized By: State Model.SM_ID

R408
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
SM to SM Comm Path carries SM to SM Event Comm
SM to SM Event Comm is carried by SM to SM Comm Path
Formalized By: SM to SM Comm Path.CPath_ID

R409
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model Event is carried to other SMs via SM to SM Event Comm
SM to SM Event Comm represents communication of State Model Event
Formalized By: State Model Event.SM_ID, State Model Event.SMevt_ID

**R410**

- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- State Model originates SM to EE Comm Path
- SM to EE Comm Path originates from State Model
- Formalized By: State Model.SM_ID

**R411**

- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- External Entity in Model has receive event communications represented by SM to EE Comm Path
- SM to EE Comm Path shows event communication to External Entity in Model
- Formalized By: External Entity in Model.EE_ID, External Entity in Model.EEmod_ID

**R412**

- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- SM to EE Comm Path carries SM to EE Event Comm
- SM to EE Event Comm is carried by SM to EE Comm Path
- Formalized By: SM to EE Comm Path.CPath_ID

**R413**

- Relationship Type: Simple
- Multiplicity/Conditionality: Mc:1
- External Entity Event is carried to EE via SM to EE Event Comm
SM to EE Event Comm represents communications of External Entity Event
Formalized By: External Entity Event.EE_ID, External Entity Event.EEevt_ID

**R414**
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1c
Imported Object represents the destination SM for SM to SM Comm Path
SM to SM Comm Path destination SM can be represented by Imported Object
Formalized By: Imported Object.IObj_ID

**R415**
Relationship Type: Subtype/Supertype
Subtypes: SM to OBJ Access Path, SM to EE Access Path
Formalized By: Access Path.APath_ID

**R416**
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model originates Access Path
Access Path originates from State Model
Formalized By: State Model.SM_ID

**R417**
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Object has data access represented by SM to OBJ Access Path
SM to OBJ Access Path shoes accesses of data from Object
Formalized By: Object.Obj_ID
**R418**

Relationship Type: Simple  
Multiplicity/Conditionality: Mc:1  
SM to OBJ Access Path carries SM to OBJ Attribute Access  
SM to OBJ Attribute Access is carried by SM to OBJ Access Path  
Formalized By: SM to OBJ Access Path.APath_ID, SM to OBJ Access Path.Obj_ID

**R419**

Relationship Type: Simple  
Multiplicity/Conditionality: Mc:1  
Attribute is accessed by SM to OBJ Attribute Access  
SM to OBJ Attribute Access represents access of Attribute  
Formalized By: Attribute.Attr_ID, Attribute.Obj_ID  
Description: None

**R420**

Relationship Type: Simple  
Multiplicity/Conditionality: Mc:1c  
Imported Object represents the destination OBJ for SM to OBJ Access Path  
SM to OBJ Access Path destination OBJ can be represented by Imported Object  
Formalized By: Imported Object.IObj_ID

**R421**

Relationship Type: Simple  
Multiplicity/Conditionality: Mc:1  
External Entity in Model has data access represented by SM to EE Access Path  
SM to EE Access Path accesses data of External Entity in Model  
Formalized By: External Entity in Model.EE_ID, External Entity in Model.EEmod_ID
R422
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
SM to EE Access Path carries SM to EE Data Item Access
SM to EE Data Item Access is carried by SM to EE Access Path
Formalized By: SM to EE Access Path.APath_ID, SM to EE Access Path.EE_ID

R423
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
External Entity Data Item is accessed by SM to EE Data Item Access
SM to EE Data Item Access represents access of External Entity Data Item
Formalized By: External Entity Data Item.EE_ID, External Entity Data Item.EEdi_ID

R424
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1c
Imported Object represents the origination SM for SM to SM Communication Path
SM to SM Communication Path origination SM can be represented by Imported Object
Formalized By: Imported Object.IObj_ID

R425
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1c
Imported Object represents the origination OBJ for Access Path
Access Path origination OBJ can be represented by Imported Object
Formalized By: Imported Object.IObj_ID
8.2.5 Subsystem ‘State Model’
Objects that have interesting behavior are given lifecycles. These lifecycles are described using State Models. A State Model consists of states, events, transactions and state actions. The State Model exists for each instance of the Object to which it is assigned. A State Model can also be an Assigner State Model of which only one can exists for all Object instances. The purpose of the Assigner State Model is to act as a single point of control through which competing requests are serialized.
Step 8: OOA of OOA
8.2.5.1 Object and Attribute Descriptions

501. State Model (SM_SM)
State Model (SM_ID, Descrip, Config_ID)
Identifier *: SM_ID

State Model.SM_ID
Full Name: State Model Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

State Model.Descrip
Full Name: State Model Description
Attribute Type: Base Attribute
Data Domain: string

State Model.Config_ID
Full Name: Configuration Identifier
Attribute Type: Base Attribute
Data Domain: unique_id
Description: The Configuration ID of the version management configuration which the state model is a part of (See Page 57 of BridgePoint Tool Guide). This ID can be used to access the V_CONFIG record corresponding to the State Model/Action Configuration.

502. State Model State (SM_STATE)
State Model State (SMstt_ID, SM_ID, SMspd_ID, Name, Numb, Final)
Identifier *: SM_ID, SMstt_ID
Identifier *2: SMspd_ID, SM_ID, SMstt_ID

State Model State.SMstt_ID
Full Name: State Model State Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

**State Model State.SM_ID**
- Attribute Type: Referential Attribute
- Refers To: State Model.SM_ID (R501) (See Page 100)

**State Model State.SMspd_ID**
- Attribute Type: Referential Attribute
- Refers To: Event Supplemental Data.SMspd_ID (R521) (See Page 112)

**State Model State.Name**
- Full Name: State Name
- Attribute Type: Base Attribute
- Data Domain: string

**State Model State.Numb**
- Full Name: State Number
- Attribute Type: Base Attribute
- Data Domain: integer

**State Model State.Final**
- Full Name: Is Deletion Final State Flag
- Attribute Type: Base Attribute
- Data Domain: integer
  - 0 = Not Deletion Final State
  - 1 = Deletion Final State

**503. State Model Event (SM_EVT)**
State Model Event (SMevt_ID, SM_ID, SMspd_ID, Numb, Mning,
  Are_KL_C, Cust_KL, Drv_Lbl, Descrip)
Identifier *: SM_ID, SMevt_ID
Identifier *2: SMspd_ID, SM_ID, SMevt_ID
State Model Event.SMevt_ID
Full Name: State Model Event Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

State Model Event.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R502)  (See Page 100)

State Model Event.SMspd_ID
Attribute Type: Referential Attribute
Refers To: Event Supplemental Data.SMspd_ID (R520)  (See Page 112)

State Model Event.Numb
Full Name: Event Number
Attribute Type: Base Attribute
Data Domain: integer

State Model Event.Mning
Full Name: Event Meaning
Attribute Type: Base Attribute
Data Domain: string

State Model Event.Are_KL_C
Full Name: Are Keyletters Custom Flag
Attribute Type: Base Attribute
Data Domain: integer
Description: This is a flag that indicates whether custom label keyletters are used for the External Entity Event.
Value 0 indicates custom label keyletters are used.
Value 1 indicates External Entity keyletters are used.

State Model Event.Cust_KL
Full Name: Custom Event Keyletters
Method

**Attribute Type:** Base Attribute

**Data Domain:** *string*

**Description:** Holds the event label - derived by concatenating the keyletters and the event number.

- If the Are_KL_C attribute is 0, then the value of the Object.Keyletter attribute is concatenated with the State Model Event.Numb attribute.
- If the Are_KL_C attribute is 1, then the value of the State Model Event.Cust_KL attribute is concatenated with the State Model Event.Numb attribute.

**State Model Event.Drv_Lbl**

Full Name: Derived Label

Attribute Type: Base Attribute

**Data Domain:** *string*

**State Model Event.Descrip**

Full Name: Event Description

Attribute Type: Base Attribute

**Data Domain:** *string*

---

**504. State Event Matrix Entry (SM_SEME)**

State Event Matrix Entry (SMstt_ID, SMEvt_ID, SM_ID, SMspd_ID)

Identifier *: SMEvt_ID, SM_ID, SMspd_ID, SMstt_ID

**State Event Matrix Entry.SMstt_ID**

Attribute Type: Referential Attribute

Refers To: State Model State.SMstt_ID (R503)  (See Page 100)

**State Event Matrix Entry.SMevt_ID**

Attribute Type: Referential Attribute

Refers To: State Model Event.SMevt_ID (R503)  (See Page 103)

**State Event Matrix Entry.SM_ID**

Attribute Type: Referential Attribute
Step 8: OOA of OOA

Refers To: State Model Event.SM_ID (R503) (See Page 101)

State Event Matrix Entry.SMspd_ID
Attribute Type: Referential Attribute
Refers To: State Model Event.SMspd_ID (R503) (See Page 102)

505. New State Transition (SM_NSTXN)
New State Transition (Trans_ID, SM_ID, SMstt_ID, SMevt_ID, SMspd_ID)
Identifier *: SM_ID, SMevt_ID, SMstt_ID, SMspd_ID
Identifier *2: Trans_ID, SM_ID, SMspd_ID

New State Transition.Trans_ID
Attribute Type: Referential Attribute
Refers To: Transition.Trans_ID (R507) (See Page 106)

New State Transition.SM_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SM_ID (R504) (See Page 103)

New State Transition.SMstt_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMstt_ID (R504) (See Page 103)

New State Transition.SMevt_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMevt_ID (R504) (See Page 103)

New State Transition.SMspd_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMspd_ID (R504) (See Page 104)
506. Event Ignored (SM_EIGN)

Event Ignored (SMstt_ID, SMevt_ID, SM_ID, SMspd_ID, Descrip)
Identifier *: SMevt_ID, SM_ID, SMstt_ID, SMspd_ID

Event Ignored.SMstt_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMstt_ID (R504)  (See Page 103)

Event Ignored.SMevt_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMevt_ID (R504)  (See Page 103)

Event Ignored.SM_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SM_ID (R504)  (See Page 103)

Event Ignored.SMspd_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMspd_ID (R504)  (See Page 104)

Event Ignored.Descrip
Full Name: Event Ignored Description
Attribute Type: Base Attribute
Data Domain: string

507. Cant Happen (SM_CH)

Cant Happen (SMstt_ID, SMevt_ID, SM_ID, SMspd_ID, Descrip)
Identifier *: SMevt_ID, SM_ID, SMstt_ID, SMspd_ID

Cant Happen.SMstt_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMstt_ID (R504)  (See Page 103)
Cant Happen.SMevt_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMevt_ID (R504) (See Page 103)

Cant Happen.SM_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SM_ID (R504) (See Page 103)

Cant Happen.SMspd_ID
Attribute Type: Referential Attribute
Refers To: State Event Matrix Entry.SMspd_ID (R504) (See Page 104)

Cant Happen.Descrip
Full Name: Cant Happen Description
Attribute Type: Base Attribute
Data Domain: string

508. Transition (SM_TXN)
Transition (Trans_ID, SM_ID, SMstt_ID, SMspd_ID)
Identifier *: Trans_ID, SM_ID
Identifier *2: Trans_ID, SM_ID, SMspd_ID

Transition.Trans_ID
Full Name: Transition Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Transition.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R505) (See Page 100)

Transition.SMstt_ID
Attribute Type: Referential Attribute
509. No Event Transition (SM_NETXN)

No Event Transition (Trans_ID, SM_ID, SMstt_ID, SMspd_ID)
Identifier *2: Trans_ID, SM_ID, SMspd_ID

No Event Transition.Trans_ID
Attribute Type: Referential Attribute
Refers To: Transition.Trans_ID (R507) (See Page 106)

No Event Transition.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model State.SM_ID (R508) (See Page 101)

No Event Transition.SMstt_ID
Attribute Type: Referential Attribute
Refers To: State Model State.SMstt_ID (R508) (See Page 100)

No Event Transition.SMspd_ID
Attribute Type: Referential Attribute
Refers To: Transition.SMspd_ID (R507) (See Page 107)

510. Creation Transition (SM_CRTXN)

Creation Transition (Trans_ID, SM_ID, SMevt_ID, SMspd_ID)
Identifier *: SM_ID, Trans_ID
Identifier *2: Trans_ID, SM_ID, SMspd_ID
Creation Transition.Trans_ID
Attribute Type: Referential Attribute
Refers To: Transition.Trans_ID (R507)  (See Page 106)

Creation Transition.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model Event.SM_ID (R509)  (See Page 102)

Creation Transition.SMevt_ID
Attribute Type: Referential Attribute
Refers To: State Model Event.SMevt_ID (R509)  (See Page 102)

Creation Transition.SMspd_ID
Attribute Type: Referential Attribute
Refers To: State Model Event.SMspd_ID (R509)  (See Page 102)

511. Moore State Model (SM_MOORE)
Moore State Model (SM_ID)
Identifier *: SM_ID

Moore State Model.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R510)  (See Page 100)

512. Mealy State Model (SM_MEALY)
Mealy State Model (SM_ID)
Identifier *: SM_ID

Mealy State Model.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R510)  (See Page 100)
513. Moore Action Home (SM_MOAH)

Moore Action Home (Act_ID, SM_ID, SMstt_ID)
Identifier *: SM_ID, SMstt_ID
Identifier *2: SM_ID, Act_ID

Moore Action Home.Act_ID
Attribute Type: Referential Attribute
Refers To: Action Home.Act_ID (R513) (See Page 110)

Moore Action Home.SM_ID
Attribute Type: Referential Attribute
Refers To: Moore State Model.SM_ID (R511) (See Page 108)

Moore Action Home.SMstt_ID
Attribute Type: Referential Attribute
Refers To: State Model State.SMstt_ID (R511) (See Page 100)

514. Mealy Action Home (SM_MEAH)

Mealy Action Home (Act_ID, SM_ID, Trans_ID)
Identifier *: SM_ID, Trans_ID
Identifier *2: SM_ID, Act_ID

Mealy Action Home.Act_ID
Attribute Type: Referential Attribute
Refers To: Action Home.Act_ID (R513) (See Page 110)

Mealy Action Home.SM_ID
Attribute Type: Referential Attribute
Refers To: Mealy State Model.SM_ID (R512) (See Page 108)

Mealy Action Home.Trans_ID
Attribute Type: Referential Attribute
Refers To: Transition.Trans_ID (R512) (See Page 106)
515. **Action Home (SM_AH)**

Action Home (Act_ID, SM_ID)
Identifier *: SM_ID, Act_ID

**Action Home.Act_ID**
Attribute Type: Referential Attribute
Refers To: Action.Act_ID (R514)  (See Page 110)

**Action Home.SM_ID**
Attribute Type: Referential Attribute
Refers To: Action.SM_ID (R514)  (See Page 110)

516. **Action (SM_ACT)**

Action (Act_ID, SM_ID, Suc_Pars, Descrip)
Identifier *: SM_ID, Act_ID

**Action.Act_ID**
Full Name: Action Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

**Action.SM_ID**
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R515)  (See Page 100)

**Action.Suc_Pars**
Full Name: Action Successfully Parsed
Attribute Type: Base Attribute
Data Domain: integer

0 = NOT Successfully Parsed
1 = Successfully Parsed
**Action.Descrip**
- Full Name: Action Description
- Attribute Type: Base Attribute
- Data Domain: `string`

**517. State Model Event Data Item (SM_EVTDI)**
State Model Event Data Item (SMedi_ID, SM_ID, Name, Descrip, DT_ID)
Identifier *: SMedi_ID, SM_ID

**State Model Event Data Item.SMedi_ID**
- Full Name: State Model Event Data Item Identifier
- Attribute Type: Base Attribute
- Data Domain: `unique_id`

**State Model Event Data Item.SM_ID**
- Attribute Type: Referential Attribute
- Refers To: State Model.SM_ID (R516) (See Page 100)

**State Model Event Data Item.Name**
- Full Name: Event Data Item Name
- Attribute Type: Base Attribute
- Data Domain: `string`

**State Model Event Data Item.Descrip**
- Full Name: Description
- Attribute Type: Base Attribute
- Data Domain: `string`

**State Model Event Data Item.DT_ID**
- Attribute Type: Referential Attribute
- Refers To: Data Type.DT_ID (R524) (See Page 29)
518. Event Supplemental Data (SM_SUPDT)

Event Supplemental Data (SMspd_ID, SM_ID)
Identifier *: SMspd_ID, SM_ID

Event Supplemental Data.SMspd_ID
Full Name: Event Supplemental Data Identifier
Attribute Type: Base Attribute
Data Domain: unique_id

Event Supplemental Data.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R523) (See Page 100)

519. Supplemental Data Items (SM_SDI)

Supplemental Data Items (SMedi_ID, SMspd_ID, SM_ID)
Identifier *: SMedi_ID, SM_ID, SMspd_ID

Supplemental Data Items.SMedi_ID
Attribute Type: Referential Attribute
Refers To: State Model Event Data Item.SMedi_ID (R522) (See Page 111)

Supplemental Data Items.SMspd_ID
Attribute Type: Referential Attribute
Refers To: Event Supplemental Data.SMspd_ID (R522) (See Page 112)

Supplemental Data Items.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model Event Data Item.SM_ID (R522) (See Page 111)

520. Instance State Model (SM_ISM)

Instance State Model (SM_ID, Obj_ID)
Identifier *: SM_ID, Obj_ID
Instance State Model.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R517) (See Page 100)

Instance State Model.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object.Obj_ID (R518) (See Page 44)

521. Assigner State Model (SM_ASM)
Assigner State Model (SM_ID, Obj_ID)
Identifier *: SM_ID, Obj_ID

Assigner State Model.SM_ID
Attribute Type: Referential Attribute
Refers To: State Model.SM_ID (R517) (See Page 100)

Assigner State Model.Obj_ID
Attribute Type: Referential Attribute
Refers To: Object.Obj_ID (R519) (See Page 44)
8.2.5.2 Relationship Descriptions

**R501**

Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model is decomposed into State Model State
State Model State ... State Model
Formalized By: State Model.SM_ID

**R502**

Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model can be communicated to via State Model Event
State Model Event ... State Model
Formalized By: State Model.SM_ID

**R503**

Relationship Type: Associative
Multiplicity/Conditionality: 1-(Mc:Mc)
State Model Event is received by State Model State
State Model State receives State Model Event
Formalized By: State Model State.SMspd_ID, State Model State.SM_ID,
State Model State.SMstt_ID, State Model Event.SMspd_ID, State
Model Event.SM_ID, State Model Event.SMevt_ID

**R504**

Relationship Type: Subtype/Supertype
Subtypes: Event Ignored, Cant Happen, New State Transition
Formalized By: State Event Matrix Entry.SMevt_ID, State Event Matrix
Entry.SM_ID, State Event Matrix Entry.SMspd_ID, State Event Matrix
Entry.SMstt_ID
**R505**
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model contains Transition
Transition ... State Model
Formalized By: State Model.SM_ID

**R506**
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model State is destination of Transition
Transition is destine to State Model State
Formalized By: State Model State.SMspd_ID, State Model State.SM_ID,
State Model State.SMstt_ID

**R507**
Relationship Type: Subtype/Supertype
Subtypes: No Event Transition, Creation Transition, New State Transition
Formalized By: Transition.Trans_ID, Transition.SM_ID,
Transition.SMspd_ID

**R508**
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model State is origination of No Event Transition
No Event Transition originates from State Model State
Formalized By: State Model State.SM_ID, State Model State.SMstt_ID

**R509**
Relationship Type: Simple
Multiplicity/Conditionality: 1c:1c
State Model Event is assigned to Creation Transition
Step 8: OOA of OOA

Creation Transition has assigned to it State Model Event
Formalized By: State Model Event.SM spd_ID, State Model Event.SM_ID,
State Model Event.SM evt_ID

R510
Relationship Type: Subtype/Supertype
Subtypes: Mealy State Model, Moore State Model
Formalized By: State Model.SM_ID

R511
Relationship Type: Associative
Multiplicity/Conditionality: 1-(1c:Mc)
State Model State ... Moore State Model
Moore State Model ... State Model State
Formalized By: Moore State Model.SM_ID, State Model State.SM_ID, State
Model State.SM stt_ID

R512
Relationship Type: Associative
Multiplicity/Conditionality: 1-(1c:Mc)
Transition ... Mealy State Model
Mealy State Model ... Transition
Formalized By: Mealy State Model.SM_ID, Transition.Trans_ID,
Transition.SM_ID

R513
Relationship Type: Subtype/Supertype
Subtypes: Moore Action Home, Mealy Action Home
Formalized By: Action Home.SM_ID, Action Home.Act_ID
R514
Relationship Type: Simple
Multiplicity/Conditionality: 1:1
Action resides in Action Home
Action Home houses Action
Formalized By: Action.SM_ID, Action.Act_ID

R515
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model contains Action
Action ... State Model
Formalized By: State Model.SM_ID

R516
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model can asynchronously communicate via State Model Event Data Item
State Model Event Data Item is carried on events into State Model
Formalized By: State Model.SM_ID

R517
Relationship Type: Subtype/Supertype
Subtypes: Instance State Model, Assigner State Model
Formalized By: State Model.SM_ID

R518
Relationship Type: Simple
Multiplicity/Conditionality: 1c:1
Object ... Instance State Model
Instance State Model ... Object
Step 8: OOA of OOA

Formalized By: Object.Obj_ID

R519
Relationship Type: Simple
Multiplicity/Conditionality: 1c:1
Object ... Assigner State Model
Assigner State Model ... Object
Formalized By: Object.Obj_ID

R520
Relationship Type: Simple
Multiplicity/Conditionality: M:1
Event Supplemental Data defines signature of State Model Event
State Model Event carries Event Supplemental Data
Formalized By: Event Supplemental Data.SMspd_ID, Event Supplemental Data.SM_ID

R521
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1c
Event Supplemental Data is delivered by received event to State Model State
State Model State receives asynchronous data via Event Supplemental Data
Formalized By: Event Supplemental Data.SMspd_ID, Event Supplemental Data.SM_ID

R522
Relationship Type: Associative
Multiplicity/Conditionality: 1-(Mc:Mc)
State Model Event Data Item makes up Event Supplemental Data
Event Supplemental Data is made up of State Model Event Data Item
Formalized By: Event Supplemental Data.SMspd_ID, Event Supplemental Data.SM_ID, State Model Event Data Item.SMedi_ID, State Model Event Data Item.SM_ID

\[ R523 \]
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
State Model contains Event Supplemental Data
Event Supplemental Data is assigned to State Model
Formalized By: State Model.SM_ID

\[ R524 \]
Relationship Type: Simple
Multiplicity/Conditionality: Mc:1
Data Type defines the type of State Model Event Data Item
State Model Event Data Item is defined by Data Type
Formalized By: Data Type.DT_ID
TASK

SW Arch Implementation
STEP 9

Develop Structural Archetypes
Step 9: Develop Structural Archetypes
An Archetype File is used as input into the Archetype Interpreter (e.g., BridgePoint Gen) - it acts as a specification of the rules by which to automatically create one or more text files. The Archetype Language controls how the file is generated. The Archetype Language is a mix of:

- literal text,
- control statements,
- and substitution variables.

The literal text is passed straight through to the output files; the control statements are used to select and iterate over data in the Generation Database; and the substitution variables are used to access from the Generation Database and format the data for the output files.

Literal Text, Control Statements, and Substitution Variables are explained in the following sections.
Step 9: Develop Structural Archetypes

9.1.1 General Language Attributes

1. Execution is sequential.

2. All transient variables are implicitly declared upon the first assignment - any subsequent assignments simply re-assign the same variable. A re-assignment of a variable to a different type is not allowed.

3. A stack execution model is assumed - variables are pushed on the stack as they are implicitly declared and are popped off the stack as they fall out of scope. Any variable implicitly declared inside of the If – Elif – Else – End if or For – End for falls out of scope when the End if / Elif / Else / End for is encountered in execution.

4. White space is treated as a token delimiter.

5. Statements are intended to be readable as a sentence so keywords are used in groups to provide verb phrases or prepositional phrases when combined with variables and OOA element references.

6. Key words may be all lower-case, all upper-case, or first character uppercase and all other characters lower-case.

7. Variables must adhere to the constrained names:
   • Names can be made up any alpha (a-z, A-Z) or numeric (0-9) characters or underscore (_) character.
   • Names cannot begin with a numeric (0-9) character.
   • Names cannot conflict with keywords from the Archetype Language.

8. Objects in the OOA of OOA are specified by using the object keyletters.

9.1.1.1 Syntax Notation

In the syntax notation used in this manual:
   • Key words and characters (operator symbols...) are in courier bold type
   • OOA element references are indicated by courier italic type
   • Variables are indicated by Palatino italic type
9.1.2 **Literal Text**

Literal text is passed straight through from the Archetype File to the generated files.

Any line in the Archetype File which is not a Control Statement is a Literal Text Line. Any line beginning with a `. .` character as the *first* non-white space character is a Control Statement Line except those lines which begin with the `..` character sequence.

A Literal Text Line with the `. .` character sequence as the *first* non-white space characters will result in the `.` character in the generated output line. The `.` character anywhere else in the Literal Text Line will result in a `.` character in the generated output line (no special treatment).

Literal Text Lines can contain Substitution Variables (see “Substitution Variables” on page 151). Substitution Variables are denoted with the `$variable` character sequence. This means that the `$` character is a special character which denotes the beginning of a Substitution Variable. The `$\$` character sequence anywhere in a literal text line will result in one `$` character in the generated output line.

Newline characters at the end of a Literal Text Line are passed through to the generated output. If you do not want a newline at the end of a generated output line (presumably due to control statement constraints), then place a `\` character as the *last* character of the Literal Text Line. The `\` character sequence as the last two characters of the Literal Text Line will result in one `\` character and one newline character as the last characters of a generated output line. The `\` character sequence as the last three characters of the Literal Text Line will result in one `\` character as the last character of a generated output line with no newline character.

**TABLE 9.1 Summary of Special Characters in Literal Text Lines**

<table>
<thead>
<tr>
<th>Character</th>
<th>Position</th>
<th>To Generate Character at Position Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>First Non-White Space</td>
<td>. .</td>
</tr>
<tr>
<td>$</td>
<td>Any</td>
<td>$$</td>
</tr>
<tr>
<td>\</td>
<td>Last</td>
<td>\</td>
</tr>
</tbody>
</table>
Step 9: Develop Structural Archetypes

9.1.3 Data Access Control Statements

The Data Access Control Statements can be any of the following:

- Instance Selection - selecting object instances from the Generation Database.
- Instance Set Iteration
- Object Attribute Access - the reading or writing of Object Attributes.

9.1.3.1 Instance Selection

Instance Selection makes direct use of chains of related object instances in the OOA of OOA Object Information Model.

The Control Language statement which supports Instance Selection is the .Select statement:

\[
\text{.Select one } \langle\text{inst_ref_var}\rangle \text{ related by } \langle\text{inst_chain}\rangle \left[ \text{ where } \langle\text{condition}\rangle \right] \\
\text{.Select any } \langle\text{inst_ref_var}\rangle \text{ related by } \langle\text{inst_chain}\rangle \left[ \text{ where } \langle\text{condition}\rangle \right] \\
\text{.Select many } \langle\text{inst_ref_set_var}\rangle \text{ related by } \langle\text{inst_chain}\rangle \left[ \text{ where } \langle\text{condition}\rangle \right] \\
\text{.Select any } \langle\text{inst_ref_var}\rangle \text{ from instances of } \langle\text{obj_keyletters}\rangle \left[ \text{ where } \langle\text{condition}\rangle \right] \\
\text{.Select many } \langle\text{inst_ref_set_var}\rangle \text{ from instances of } \langle\text{obj_keyletters}\rangle \left[ \text{ where } \langle\text{condition}\rangle \right]
\]

\text{where:}
\begin{align*}
\langle\text{inst_ref_var}\rangle & : = \text{reference to 0 or 1 object instances} \\
\langle\text{inst_ref_set_var}\rangle & : = \text{reference to 0, 1, or more object instances} \\
\langle\text{inst_chain}\rangle & : = \text{list of object, relationship, object... which form an unbroken path through a series of related object instances} \\
\langle\text{obj_keyletters}\rangle & : = \text{keyletters of an object} \\
\langle\text{condition}\rangle & : = \text{expression with boolean result (see “Expressions” on page 145)}
\end{align*}
Here are some examples:

To select all instances of objects from the OOA:

```
.Select many obj_set from instances of O_OBJ
```

To select all attribute instances related to an object instance obj_inst:

```
.Select many attr_set related by obj_inst->O_ATTR[R102]
```

To select all relationships which an object instances is involved in:

```
.Select many rel_set related by obj_inst->R_OIR[R201]->[R_REL[R201]
```

Note that the navigation through the associative relationship R201 was in 2 steps - first to the associative object and then to the other side of the associative relationship.

The resulting <inst_ref_var>/<inst_ref_set_var> is a transient variable which follows the implicit declaration rule. When the resulting <inst_ref_var>/<inst_ref_set_var> is being implicitly declared (used for the first time), the referred to object of the transient variable is set according to the result of the .Select. When the resulting <inst_ref_var>/<inst_ref_set_var> is being reassigned, the referred to object of the new selection must match that of the transient variable.

### 9.1.3.2 Instance Chains

The related by form of the .Select statement uses an instance chain to specify a path through the related instances. An instance chain is simply a chain of object instances which are related through the specified relationships - the eventual result is 0, 1, or more instances of the last object of the chain. The syntax of the instance chain places the focus on the objects of the chain (specified by the object keyletters) because the instances of the chain are object instances. The \([ ]\) syntax is intended to indicate access into a table of that object's instances. The contents of the \([ ]\) is a specification of which instances are being accessed - since the instances are accessed via a relationship, the contents of the \([ ]\) is the relationship traversal specification.

The relationship traversal specification can be specified as:
Step 9: Develop Structural Archetypes

R<number>

or

R<number>.<direction>

where:

R<number> ::= reference to relationship

<direction> ::= specification of the direction of the traversal -

IR for ID side to Referential side and RI for Referential side to ID side.

It is suggested that you use the simplest expression of the relationship traversal direction which does not lead to ambiguity.

The <direction> is needed when traversing reflexive relationships, i.e., relationship where objects are related to themselves, since reflexive relationships can be traversed in each direction. Examples of reflexive relationships in the OOA of OOA are R103 (to specify order of attributes) and R112 (to specify order of R#'s). For example:

.Select one prev_attr_inst related by curr_attr_inst->O_ATTR[R103.RI]

Selects the previous attribute instance - the reason RI is used is because PAttr_ID (Previous Attribute ID) is used to formalize the relationship R103 - this means that the selection will find the instance of O_ATTR in which selected.Attr_ID is equal to curr_attr_inst.PAttr_ID which is the previous attribute in respect to curr_attr_inst.

9.1.3.3 Chain Multiplicity & Conditionality

The multiplicity of an instance chain is zero or one (one) if the starting instance variable has multiplicity zero or one and all relationship traversals in the chain result have multiplicity zero or one. Otherwise, the multiplicity of the instance chain is zero, one, or more (many).

One can only be used with an instance chain of multiplicity zero or one, whereas any and many can only be used with an instance chain of multiplicity zero, one, or many.
The conditionality of an instance chain is unconditional if all relationship traversals in the chain are unconditional; otherwise, the instance chain is conditional.

9.1.3.4 Where Clause

The **where (condition)** can be used to efficiently filter out a subset of the instances selected through the **from instances of** or **related by** constructs. The **condition** is applied separately to each object instance selected through the **from instances of** or **related by** constructs - the instances for which **condition** is TRUE are selected - the instances for which **condition** is FALSE are not selected.

**<condition>** is a boolean expression (see “Expressions” on page 145) - the current instance being selected is referred to by the keyword **selected**. Here are some examples.

To select all attributes named Id:

```
.Select many attr_set from instances of O_ATTR
  where (selected.name == "Id")
```

To select all attributes in objects with keyletters DOG:

```
.Select many attr_set from instances of O_ATTR
  where ("${selected ->O_OBJ[R102]}.key_lett"
  == "DOG")
```

Note: The preceding example uses an instance substitution variable in a quoted string (see “Quoted Strings” on page 144) and an instance chain within the substitution variable (see “Substitution Variables” on page 151).

9.1.3.5 Instance Set Iteration

Once a set of instances has been selected, the template designer may want to specify statements to be carried out on each one of the instances of the set. The Control Statement which supports this is:
Step 9: Develop Structural Archetypes

.For each <inst_ref_var> in <inst_ref_set_var>

<stmt_blck>

.End for

where:

<inst_ref_var> ::= reference to 1 object instance
<inst_ref_set_var> ::= reference to 0, 1, or more object instances
<stmt_blck> ::= block of Archetype Language statements

The statements in the .For structure are executed once for each instance in the set. The iterations are sequential in a repeatable order, i.e., the order of the instances in a set are consistent from one execution to another. For example:

Start
 .Select many obj_set from instances of O_OBJ
 .For each obj_inst in obj_set
       Object name is ${obj_inst.name}
 .End for
 Finish

will result in the name of each object being printed on a separate line in the generated output. Each time the above example is executed, the order of the object names will be the same.

Note: When instances are added to the generation database, the order of elements is implementation specific when the order is compared to the order before the instances where added.

The variable <inst_ref_var> is scoped within the <stmt_blck>, i.e., it out of scope after the .End for. However, if the scope of <inst_ref_var> needs to extend beyond the .End for, then define <inst_ref_var> prior to the .For statement. In the previous example, obj_inst is out of scope (and no longer on the stack) when the 'Finish' Literal Text Line is encountered. In the following example, obj_inst is still in scope (and still on the stack) when the 'Finish' Literal Text Line is encountered:

Start
 .Select any obj_inst from instances of O_OBJ

End
Method

.Select many obj_set from instances of 0_OBJ
.For each obj_inst in obj_set
    Object name is ${obj_inst.name}
.End for
Object name is ${obj_inst.name}
Finish

Since all instances are ordered and therefore, iteration through instance sets is sequential, the following statement is provided to break out of the iteration through the ordered set, presumably because you have found what you were looking for:

.Break for

9.1.3.6 While

The while statement provides a general purpose iteration mechanism. This complements the other iteration mechanism, the for each statement. The for each statement is a specific purpose iteration mechanism to iterate through an object instance reference set. The while statement is a general purpose iteration mechanism to iterate until the while condition is false. The syntax of the while statement is:

.while (<boolean expression>)
    <statement>
    <statement>
    ...
    <statement>
.end while

The statements between the while and end while will be executed in sequence until the <boolean expression> is false. The condition is checked before the first iteration.
A break while statement is available, providing an alternative technique to end the iteration. The syntax of the break while statement is:

```
while (<boolean expression>)
    <statement>
    <statement>
    break while
    ...
    <statement>
.end while
```

When executed, the break while statement will cause control to be transferred to the statement after the end while corresponding to the innermost executing while statement. For example:

```
assign count = 1
    while (count < 10)        // while1
        .while (1 == 1)        // while2
            .if (<condition>)
                .break while  // break2
            .end if
        .end while              // endwhile2
    .if (<condition2>)
        .break while        // break1
    .end if
.end while            // endwhile1
```

Execution of 'break2' will cause control to transfer to the statement following 'endwhile2'. Execution of 'break1' will cause control to transfer to the statement following 'endwhile1'.

---

**Step 9: Develop Structural Archetypes**
9.1.3.7 Object Attribute Access

The form of an Attribute Access in the Archetype Language is:

\[
<\text{obj\_inst\_ref\_var}> . <\text{attribute}>
\]

where:

\[
<\text{obj\_inst\_ref\_var}> :: \text{variable holding a handle to 1 object instance} \\
<\text{attribute}> :: \text{name of object attribute}
\]

*Great care should be taken when writing object attributes.* Writing an object attribute will permanently affect the value for all future uses of the attribute... - the new value of the attribute is stored persistently in the generation database.

9.1.4 Transformer Control Statements

The transformer provides for computational logic.

9.1.4.1 Assign Statement

The `.Assign` statement makes use of Expressions (See “Expressions” on page 145).

The `.Assign` statement has the following syntax:

\[
\text{.Assign} \ <\text{variable}> = \ <\text{expression}>
\]

where:

\[
<\text{variable}> :: \text{data item, e.g., object attribute, fragment attribute, or transient variable.} \\
<\text{expression}> :: \text{expression - usually a calculation using object attribute access, literal values, ...}
\]

When `<variable>` is an object attribute, the data type of `<expression>` must be compatible with the data type of `<variable>` (See Table 9.2 below).
If `<variable>` is a transient variable, then that transient variable follows the implicit declaration rule. When a transient variable is being implicitly declared (assigned for the first time), the data type of the transient variable is set to be the same as the data type of the `<expression>`. When a transient variable is being re-assigned, the data type of `<expression>` must be compatible with the data type of the `<variable>` (See Table 9.2 below).

**TABLE 9.2 Compatible Assignment Data Types**

<table>
<thead>
<tr>
<th><code>&lt;variable&gt;</code> Data Type</th>
<th><code>&lt;expression&gt;</code> Data Type</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>real</td>
<td>real</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>real</td>
<td>Truncates all digits after the decimal point.</td>
</tr>
<tr>
<td>real</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><code>inst_ref&lt;Object&gt;</code></td>
<td><code>inst_ref&lt;Object&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>inst_ref_set&lt;Object&gt;</code></td>
<td><code>inst_ref_set&lt;Object&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>frag_ref</code></td>
<td><code>frag_ref</code></td>
<td></td>
</tr>
</tbody>
</table>

If `<variable>` is of data type `inst_ref<Object>`, `inst_ref_set<Object>`, or `frag_ref<Object>`, then `<expression>` may be one of the following:

- Transient Variable
- Fragment Attribute

Here are some examples:

- Assign `obj_inst = prev_obj_inst`
- Assign `obj_set = next_obj_set`
- Assign `attr_inst = base_attr_frag.base_attr_inst`
- Assign `data_type_frag = attr_data_type_frag`
9.1.5 Tester Control Statements

Testers are supported in the Archetype Language with the `.if` statement:

```
.if (<condition>)
  <stmt_blk>
[.elif (<condition>)
  <stmt_blk>]
[.else
   <stmt_blk>]
.end if
```

*where:*

- `<condition>` ::= expression with **boolean** result (see “Expressions” on page 145)
- `<stmt_blk>` ::= block of Archetype Language statements

Many `elif` (else if) constructs may be present in the same `if` construct.

Here are some examples:

```
.if (obj_inst.numb < 100)
   literal text...
.elif ((obj_inst.numb >= 200) && (obj_inst.numb < 300))
   literal text...
.else
   literal text...
.end if
```

```
.if ("${obj_inst.descrip:TASK}" == "CLIENT")
   source code for client...
.elif ("${obj_inst.descrip:TASK}" == "SERVER")
```

source code for server...
.

.else
  .print "Error in specification of obj_inst.descrip:TYPE"
  .exit 1
.end if

.Assign min_state_num = 999999
.Select any min_state_num_inst from instances of SM_STATE
.Select many state_set related by sm_inst->SM_STATE[R501]
.For each state_inst in state_set
  .If (state_inst.numb == min_state_num)
    .print "OOA Data NOT Audited - 2 states with same numb"
  .else if (state_inst.numb < min_state_num)
    .Assign min_state_num_inst = state_inst
  .end if
.end for

9.1.6 Function Control Statements

Functions are supported in the Archetype Language to allow reuse of blocks of Archetype Language Statements. Functions always return a fragment - a fragment is a small piece of generated output. The intent of functions is to use them to build fragments which can be plugged into larger fragments and eventually into the whole generated file.

To define a function, use the .Function statement:

    .Function <function_name>
      [.Param <param_type> <param_name>
      .Param <param_type> <param_name>
    .end Function
Method

...]
<stmt_blck>]
.end function

where:
<function_name> ::= the name of the function
<param_type> ::= the type of the parameter - allowed types are:
boolean
integer
real
string
inst_ref
inst_ref_set
frag_ref
<param_name> ::= the name of the parameter
<stmt_blck> ::= block of Archetype Language statements - includes Literal Text, Control Statements, and Substitution Variables

To invoke a function, use the .Invoke statement:

.Invoke [ <frag_ref_var> = ] <function_name>
(<actual_param>, <actual_param>...)

where:
<frag_ref_var> ::= a transient variable which holds a reference to a fragment
<function_name> ::= the name of the function being invoked
<actual_param> ::= actual parameter (See Table 9.3 below)

9.1.6.1 Fragment Attributes

Attributes may be defined for a fragment when the fragment is formed with the function invocation. The attribute body is always defined - after invocation of
Step 9: Develop Structural Archetypes

A function, body will contain the output generated from the Literal Text Lines in the function.

Additional attributes are defined by declaring transient variables inside the function of the form:

\[\text{attr}_\text{xxx}\]

For example:

```plaintext
.Function get_attr_type
.Param inst_ref p_attr_inst
.Assign attr_type = "${p_attr_inst.descrip:TYPE}"
.End function
```

will result in the variable \text{type} being available for use through the fragment reference returned from the invocation of the function:

```plaintext
.Select any dog_inst from instances of O_OBJ where (selected.name == "Dog")
.Select any dog_attr_inst related by dog_inst->O_ATTR[R012]
.Invoke dog_attr_type = get_attr_type (dog_attr_inst)
```

The type of the attribute

\[${\text{dog_inst.name}}.\{\text{dog_attr_inst.name}\}\]

is \[${\text{dog_attr_type.type}}\].

Be careful to make sure the \text{attr}_\text{xxx} variables are in scope when the \text{.End function} is reached. For example:

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Actual Parameter Forms Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Rvalue of type \text{boolean}</td>
</tr>
<tr>
<td>integer</td>
<td>Rvalue of type \text{integer}</td>
</tr>
<tr>
<td>real</td>
<td>Rvalue of type \text{real}</td>
</tr>
<tr>
<td>string</td>
<td>Rvalue of type \text{string}</td>
</tr>
<tr>
<td>inst_ref</td>
<td>&lt;transient_var&gt; of type \text{inst_ref}</td>
</tr>
<tr>
<td>inst_ref_set</td>
<td>&lt;transient_var&gt; of type \text{inst_ref_set}</td>
</tr>
<tr>
<td>frag_ref</td>
<td>&lt;transient_var&gt; of type \text{frag_ref}</td>
</tr>
</tbody>
</table>
Method

.Function get_attr_type
.Param integer p_value
.If (p_value < 100)
    .Assign attr_new_value = 22
.Else
    .Assign attr_new_value = 2000;
.End if
.End function

results in the transient variable attr_new_value NOT to become a fragment attribute since it falls out of scope with the .If statement and therefore, is not on the stack when the .End function is encountered. A way to handle this case is:

.Function get_attr_type
.Param integer p_value
.Assign attr_new_value = 0
.If (p_value < 100)
    .Assign attr_new_value = 22
.Else
    .Assign attr_new_value = 2000;
.End if
.End function

9.1.7 File Control Statements

File Control Statements in Archetype Language allow management of the Archetype Files.

9.1.7.1 Emitting Generated Output
All generated output is buffered as it is generated from interpretation of Literal Text Lines. To transfer the output from the buffer to a file, use:

`.Emit to file "<file_name>"

The `.Emit` also clears the buffer’s contents. For example:

```
.Emit to file "/source_code/$_{ss_inst.name}/$_{obj_inst.name}.cpp"
```

will result in a file being generated in a directory based on the subsystem name with a name based on the object name.

If a generated file is emitted and a file of the same name already exists, then the newly generated file is compared to the existing file - if the files are the same, then the existing file is left undisturbed (so that modification times... are left intact) - if the files are different, then the existing file is replaced with the newly generated file.

To clear the contents of the buffer without emitting the contents to a file, use:

`.Clear`

### 9.1.7.2 Comments

To add a comment in the Archetype File, use:

```
.Comment <user_comment>
or
.// <user_comment>
```
At least one white space character must follow the .Comment keyword. A white space character does not need to follow the .// keyword. All text to the end of the line after the comment keyword is ignored.

9.1.7.3 Include

To include another Archetype File, use:

```
.Include "<file_name>"
```

When a file is included, a marker is placed on the stack and the interpreter begins interpretation on the first line of the included file. When all lines in the included file have been processed, then all variables pushed on the stack since the include marker was pushed on the stack are considered out of scope (and therefore popped from the stack) - the interpreter then resumes interpretation on the line following the .Include statement.

9.1.7.4 Handling Errors

In handling errors from your Archetype Files, use:

```
.Print "<error_message>"
```

to print a message to stderr about the problem which was found and use:

```
.Exit <exit_status>
```

to stop the interpreter with integer value <exit_status>.

9.1.8 Rvalues

An Rvalue is a specification of a literal value or the specification of a variable.
9.1.8.1 Literals as Rvalues

Literal values must be able to be entered for the Core Data Types. Table 9.4 below uses example specifications to illustrate how the literal values are specified.

TABLE 9.4 Literal Specification for Core Data Types

<table>
<thead>
<tr>
<th>Core Data Type</th>
<th>Literal Specification Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
<tr>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>-10</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td>real</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>256.44</td>
</tr>
<tr>
<td></td>
<td>-10.3</td>
</tr>
<tr>
<td></td>
<td>3.1415</td>
</tr>
<tr>
<td>string</td>
<td>“Hello World”</td>
</tr>
<tr>
<td>inst_ref&lt;Object&gt;</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>{inst_ref&lt;Object&gt;}</td>
<td>Not Allowed</td>
</tr>
<tr>
<td>frag_ref</td>
<td>Not Allowed</td>
</tr>
</tbody>
</table>

9.1.8.2 Quoted Strings

Quoted strings get special handling in the Archetype Language - each Quoted String is treated as a Literal Text Line and is run through the variable substituter. For example:

```
.Assign name = dog_inst.name
```

and

```
.Assign name = "${dog_inst.name}"
```
are equivalent. Treating Quoted Strings as Literal Text Lines adds flexibility in concisely specifying the string value. For example, the following shows Substitution Variables used in the .if statement and .emit statement:

```
.Select many obj_set from instances of O_OBJ
.For each obj_inst in obj_set
   .if ("${obj_inst.descrip:CLIENT_SERVER}" == "CLIENT")
      Output statements for client here...
   .end if
   .emit to file "${obj_inst.key_lett}.cc"
.end for
```

Since the Quoted Strings get run through the Literal Text Substituter, use $$ to yield one $ character.

In addition, use " " to yield one " character.

### 9.1.8.3 Variables as Rvalues

The variables of the following types may be used as values:

- `<transient_variable>` of type `boolean, integer, real, or string`
- `<inst_ref_var> . <attribute>` where `<attribute>` is of type `boolean, integer, real, or string`
- `<frag_ref_var> . <attribute>` where `<attribute>` is of type `boolean, integer, real, or string`

### 9.1.9 Expressions

The Archetype Language supports simple expressions and compound expressions.

#### 9.1.9.1 Simple Expressions

`Simple` expressions are single unary or binary operations:
Step 9: Develop Structural Archetypes

Here are some examples:

```
if (empty obj_inst)
assign number_selected = cardinality obj_set
if (obj_inst.numb >= 100)
assign attr_decl = "${attr_inst.descrip:TYPE}
$cr{attr_inst.name};"
```

### 9.1.9.2 Compound Expressions

Simple expressions can be combined to form a compound expressions:

```
(<unary_operator> <expression>)
(<expression> <binary_operator> <operand>)
(<operand> <binary_operator> <expression>)
(<expression> <binary_operator> <expression>)
```

where:

- `<unary_operator>` ::= unary operator
- `<binary_operator>` ::= binary operator
- `<operand>` ::= operand, e.g., literal value, object attribute, or transient variable
- `<expression>` ::= expression - either a simple expression or a compound expression
Note the *required* use of the ( and ) characters to delimit expressions in a compound expression. This takes away the issues surrounding precedence and associativity of operators.

Here are some examples:

```plaintext
.if ((x > 1) AND (x < 10))
.assign full_name = ((first_name + middle_name) + last_name)
.select many processor1_objs from instances of O_OBJ where (("${selected.descrip:TASK}" == "TASK1") OR ("${selected.descrip:TASK}" == "TASK4"))
```

### 9.1.9.3 Operations

#### TABLE 9.5 Core Unary Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>Logical Negation</td>
</tr>
<tr>
<td>empty</td>
<td>inst_ref&lt;Object&gt; or (inst_ref&lt;Object&gt;) test for empty set</td>
</tr>
<tr>
<td>not_empty</td>
<td>inst_ref&lt;Object&gt; or (inst_ref&lt;Object&gt;) test for not empty set</td>
</tr>
<tr>
<td>first</td>
<td>Test if the (inst_ref&lt;Object&gt;) internal cursor is on the first in the set</td>
</tr>
<tr>
<td>not_first</td>
<td>Test if the (inst_ref&lt;Object&gt;) internal cursor is not on the first in the set</td>
</tr>
<tr>
<td>last</td>
<td>Test if the (inst_ref&lt;Object&gt;) internal cursor is on the last in the set</td>
</tr>
<tr>
<td>not_last</td>
<td>Test if the (inst_ref&lt;Object&gt;) internal cursor is not on the last in the set</td>
</tr>
<tr>
<td>cardinality</td>
<td>Count of items in (inst_ref&lt;Object&gt;)</td>
</tr>
</tbody>
</table>
TABLE 9.6 Core Binary Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>Logical And</td>
</tr>
<tr>
<td>or</td>
<td>Logical Inclusive Or</td>
</tr>
<tr>
<td>+</td>
<td>Arithmetic Addition (\texttt{integer} &amp; \texttt{real}) or Concatenation (\texttt{string})</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Arithmetic Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Quotient from Arithmetic Division</td>
</tr>
<tr>
<td>%</td>
<td>Remainder from Arithmetic Division</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less-than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less-than or Equal-to</td>
</tr>
<tr>
<td>==</td>
<td>Equal-to</td>
</tr>
<tr>
<td>!=</td>
<td>Not-equal-to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater-than or Equal-to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater-than</td>
</tr>
</tbody>
</table>

TABLE 9.7 Core Unary Operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>empty</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>not_empty</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>empty</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>not_empty</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>first</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>not_first</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>last</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>not_last</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>boolean</td>
</tr>
<tr>
<td>cardinality</td>
<td>\texttt{inst_ref\langle Object\rangle}</td>
<td>integer</td>
</tr>
</tbody>
</table>
### TABLE 9.8 Core Binary Operations

<table>
<thead>
<tr>
<th>Left Operand</th>
<th>Operator</th>
<th>Right Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>and or == !=</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>integer</td>
<td>+ - * / %</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>integer</td>
<td>&lt; &lt;= == != &gt;= &gt;</td>
<td>integer</td>
<td>boolean</td>
</tr>
<tr>
<td>integer</td>
<td>+ - * /</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>integer</td>
<td>&lt; &lt;= == != &gt;= &gt;</td>
<td>real</td>
<td>boolean</td>
</tr>
<tr>
<td>real</td>
<td>+ - * /</td>
<td>integer</td>
<td>real</td>
</tr>
<tr>
<td>real</td>
<td>&lt; &lt;= == != &gt;= &gt;</td>
<td>integer</td>
<td>boolean</td>
</tr>
<tr>
<td>real</td>
<td>+ - * /</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>real</td>
<td>&lt; &lt;= == != &gt;= &gt;</td>
<td>real</td>
<td>boolean</td>
</tr>
<tr>
<td>string</td>
<td>+</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>string</td>
<td>&lt; &lt;= == != &gt;= &gt;</td>
<td>string</td>
<td>boolean</td>
</tr>
</tbody>
</table>

### TABLE 9.9 Set Operations

<table>
<thead>
<tr>
<th>Left Operand</th>
<th>Operator</th>
<th>Right Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>&lt;inst_ref&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>&amp; &lt;inst_ref&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>- &lt;inst_ref&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>== &lt;inst_ref&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>!= &lt;inst_ref&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>&lt; &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>&amp; &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref&gt;</td>
<td></td>
<td>- &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td></td>
<td>&lt; &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td></td>
<td>&amp; &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td></td>
<td>- &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td></td>
<td>&lt; &lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
</tbody>
</table>
### TABLE 9.9 Set Operations

<table>
<thead>
<tr>
<th>Left Operand</th>
<th>Operator</th>
<th>Right Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td>&amp;</td>
<td>&lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td>-</td>
<td>&lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td>==</td>
<td>&lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
<tr>
<td>&lt;inst_ref_set&gt;</td>
<td>!=</td>
<td>&lt;inst_ref_set&gt;</td>
<td>&lt;inst_ref_set&gt;</td>
</tr>
</tbody>
</table>
9.1.10 Substitution Variables

Literal text lines can contain substitution variables which allow information to be pulled out of the Generation Database and be placed into the generated files.

A substitution variable takes on the following form:

$ <format> \{ <inst_ref_var> . <attribute> [ : <parse_keyword> ] \}$

or

$ <format> \{ <inst_chain> . <attribute> [ : <parse_keyword> ] \}$

or

$ <format> \{ <frag_ref_var> . <attribute> \}$

or

$ <format> \{ <transient_var> \}$

where:

$ <format> ::=$ represents instructions on how to format the string which is substituted into the generated file

$ <inst_ref_var> ::=$ reference to an object instance from the OOA of OOA

$ <inst_chain> ::=$ an instance chain which results in one instance (see "Instance Chains" on page 129)

$ <frag_ref_var> ::=$ reference to a fragment which has been returned from a function

$ <attribute> ::=$ attribute of the object referred to by $<inst_ref_var>$ or attribute of the fragment referred to by $<frag_ref_var>$

$ <parse_keyword> ::=$ represents a keyword which can be parsed for in the string which is substituted into the generated file

Here are some examples:

$\{obj\_inst.name\}$

$\{ss\_inst.name\}$

$\{dt\_inst.descrip:TYPE\}$

$\{attr\_inst->O\_OBJ[R102].key\_lett\}$

$\{rattr\_inst->O\_BATTR[R113]->O\_ATTR[R106].name\}$
9.1.10.1 Format

<format> is needed to allow the legal names in OOA to be transformed into legal names in the generated file. For example, spaces are allowed in object names in OOA but are not allowed in class names in C++ - if the object name from the OOA is to be used as the class name in a generated C++ file, then the object name must be transformed into a legal C++ name.

The <format> characters allowed are listed in Table 9.10 below.

<table>
<thead>
<tr>
<th>Format Character</th>
<th>Format Affect</th>
</tr>
</thead>
<tbody>
<tr>
<td>U or u</td>
<td>Upper - make all characters upper case</td>
</tr>
<tr>
<td>C or c</td>
<td>Capitalize - make the first character of each word capitalized and all other characters of a word lower case</td>
</tr>
<tr>
<td>L or l</td>
<td>Lower - make all characters lower case</td>
</tr>
<tr>
<td></td>
<td>Underscore - change all white space characters to underscore characters</td>
</tr>
<tr>
<td>R or r</td>
<td>Remove - remove all white space</td>
</tr>
<tr>
<td></td>
<td>Note: The removal of white space will occur after the capitalization has taken place in the case of the CR or RC combination.</td>
</tr>
</tbody>
</table>

Here are some examples:

<table>
<thead>
<tr>
<th>Input</th>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective Spectrum</td>
<td>u</td>
<td>OBJECTIVE SPECTRUM</td>
</tr>
<tr>
<td>Objective Spectrum</td>
<td>u_</td>
<td>OBJECTIVE_SPECTRUM</td>
</tr>
<tr>
<td>Objective Spectrum</td>
<td>ur</td>
<td>OBJECTIVESPECTRUM</td>
</tr>
<tr>
<td>ObjectIVE SpecTRum</td>
<td>c</td>
<td>Objective Spectrum</td>
</tr>
<tr>
<td>ObjectIVE SpecTRum</td>
<td>c_</td>
<td>Objective_Spectrum</td>
</tr>
<tr>
<td>ObjectIVE SpecTRum</td>
<td>cr</td>
<td>ObjectiveSpectrum</td>
</tr>
<tr>
<td>ObjectIVE SpecTRum</td>
<td>l</td>
<td>objective spectrum</td>
</tr>
<tr>
<td>ObjectIVE SpecTRum</td>
<td>l_</td>
<td>objective_spectrum</td>
</tr>
<tr>
<td>ObjectIVE SpecTRum</td>
<td>lr</td>
<td>objectivespectrum</td>
</tr>
</tbody>
</table>
9.1.10.2 Parse Keyword

The *parse-keyword* is used to facilitate simplified file generation through avoiding ‘expanding’ the OOA of OOA. Rather than adding ‘expansion’ objects which are related to OOA of OOA objects to capture design information, the *parse-keyword* can be placed directly in the OOA capture and what follows the *parse-keyword* until the next new-line character is available with the $format{instance-ref.attribute:parse-keyword} construct. For example, if an object description contains:

```
This attribute captures the name of the quick brown fox
who jumped over the lazy brown dog.
```

**TYPE**: String

**LENGTH**: 64

then type TYPE can be pulled out with:

```
${attr_inst.descrip:TYPE}
```

and the LENGTH can be pulled out with:

```
${attr_inst.descrip:LENGTH}
```

Note that the above example explicitly places *design/implementation* information into the *analysis* - this has ramifications on the reusability of the analysis across different designs and implementation technologies. Use with care!

9.1.10.3 Information Substitution Variables

There are some special substitution variables available which can be used anywhere:

```
${info.date}
${info.user_id}
${info.arch_file_name}
${info.arch_file_line}
```
Obviously, \texttt{info} is a keyword and can not be used as a transient variable name. \texttt{$\{\text{info.unique_num}\}$} generates a unique integer each time it is referenced. For example, the first time it is referenced, it may produce 1, the next time 2, the next time 3... The order of the unique numbers generated will be exactly the same from one invocation of the Archetype Interpreter to the next.
9.2 Automation

9.2.1 Overview
Files can be automatically generated by using an archetype file and an archetype interpreter as shown in Figure 9.2.1.1.

Figure 9.2.1.1. Archetype Interpretation Architecture

An archetype file is used as input into the archetype interpreter - it acts as a specification of the rules by which to automatically create one or more text files. The archetype language controls how the file is generated. The archetype language is a mix of literal text, control statements, and substitution variables.
Step 9: Develop Structural Archetypes

Generally, one file archetype file exists for each type of file which is generated. For example, for C++ code generation, one file archetype file exists for header files, one for the source files, and one for make files.

The **xxx.gen** file is a generation database being built by the *gen_import* command. Note that the **xxx.gen** file must reside physically on the BridgePoint server similar to analyst .ooa files.

### 9.2.2 Running *gen_import*

*gen_import* must be run at least twice to create the generation database for translation. The first run of *gen_import* will be with provided **ooa_schema.sql** file to build all the necessary TABLES to support accessing data from any object in the OOA of OOA or accessing data via any relationship in the OOA of OOA. The format of the first *gen_import* command is:

```
gen_import database .gen ooa_schema.sql
```

The second run of *gen_import* will populate the generation database with the data from the application OOA. You must export data from analyst using the Export-SQL option into a file ending with the .sql suffix. Do not leave the Include Graphical Data check box checked. Graphical data has no role in most translation you are trying to translate the graphical data into another form. The format of the second *gen_import* is:

```
gen_import [-d num] database .gen data_file .sql
```

Note that the -d option is for specifying domain code between 0-15 inclusive; if domain code is not provided, a unique one will be chosen/allocated for this import.

You are now ready to generate code.

### 9.2.3 Running *gen_file*

The **database .gen** file is the generation database created by the *gen_import* command. The **archetype .arc** file contains the archetype for a portion of the target architecture.

There are several options on the *gen_file* command.
The “-# num” allows the generation of a specified number of output files for the archetype. For example, if an archetype specifies the generation of one file per object, potentially many files are generated from one archetype. This option allows control in order to generate a limited number during a development phase.

The “-q” option will cause gen_file to quit after the first error. Without this option, an incorrectly specified .arc can iteratively generate errors.

The “-f file” allows the generation of a specific file.

The “-l” option specifies that the output will be directed to a file with the name database.gen_log.

The usage of gen_file is:

\[
\text{gen\_file [ -# num ] [-l] [-q] [-f \text{file}] database.gen archetype.arc}
\]

You simply write an archetype file and run it through the gen_file process and it generates files per the instructions!

9.2.4 Using Makefiles

Invocations of gen_import and gen_file can be placed in makefiles to fully automate the process of File Generation. Here are some tips:

- The exit status of gen_import and gen_file is the number of errors which have occurred during execution. Make runs smoothly as long as the exit status of each command is 0 - if the exit status of a command is not 0, then make stops execution of the commands for that rule... Therefore, your make will respond when the errors have occurred during execution of gen_import or gen_file.

- In gen_file, the Control Statement \'.exit <exit_status>\' causes the gen_file to exit with the exit status <exit_status> - this can allow the user to get make to respond upon errors found in the generation.
STEP 10

*Develop Action Archetypes*
Step 10: *Develop Action Archetypes*
10.1 **Method**

Action Specifications are specified in the Action Language within the Application & Service OOAs. The Actions must be converted to the Target Source Code. This is accomplished through Action Generation.

Action Source Code is generated using a technique very similar to that used by a compiler to produce machine code from a high level language. However, one major difference exists: the last step of the Action Generation is done through a set of user-specified Fragment Generation Functions so that the user has complete control over the process of generation and therefore can generate source code in any Programming Language (C, C++, Smalltalk, Fortran, COBOL, Lisp, Assembler, 4GL...) and any Software Architecture.
The process is of Action Generation is:

The steps of the process are:

1. **Script Generation** - break Action Language into fundamental components - fragments - from the inside out, build smallest fragments first, combine into larger fragments, finally yield one resulting fragment.

2. **Action Source Code Generation** - plug in user definitions for each fragment - will generate small fragments of Target Source Code, combine the small fragments into larger fragments of Target Source Code, and finally yield the resulting Target Action Source Code.
10.1.1 Invoking Fragment Generation

For Script Generation only:

```
.AL_xlate <action_location> <action_inst_ref> script to file
  "<file_name>"
```

where:

- `<action_location>` ::= keyword either `instance_sm` or `assigner_sm`.
- `<action_inst_ref>` ::= Variable of type `inst_ref` which holds a reference to the an instance of the OOA of OOA object SM_ACT.
- `<file_name>` ::= Specification of file for output of Fragment Generation Script

For Script Interpretation:

```
.Include "<file_name>"
```

where:

- `<file_name>` ::= Specification of file for output of Fragment Generation Script

For both Script Generation and Script Interpretation:

```
.AL_xlate <action_location> <action_inst_ref>
```

where:

- `<action_location>` ::= keyword either `instance_sm` or `assigner_sm`.
- `<action_inst_ref>` ::= Variable of type `inst_ref` which holds a reference to the an instance of the OOA of OOA object SM_ACT.
For example, a simple File Archetype using Action Generation may look like:

*Include “Frag-gen.arc”*

Select many state_models from instances of SM_ISM

For each state_model in state_models

Select many states related by state_model -> SM_SM

[SM_STATE [R301]]

For each state in states

OS_$(state.name) ()

{  
  Select one action related by state -> SM_MOAH
  [SM_AH [R513] ->SM_ACT [R514]]
  AL_xlate instance_sm action
}

End for

Emit to file “{$state_model.name}.cpp”

End for

### 10.1.2 Fragment Generation Script

The strategy behind fragment generation is to begin with the innermost statement components and convert them to Target Source Code, and then move to combining small fragments into larger fragments... until a statement is formed - statements are combined into blocks & blocks into actions...

For example, lets examine the following Action Language statement and generation of equivalent C code:

\[
Assign x = rcvdEvt.a + 2;
\]
We begin by building a fragment for \texttt{rcvdEvt.a}:

\begin{verbatim}
get_evt_data_item_a ()
\end{verbatim}

Next, build a fragment for the literal integer value 2:

\begin{verbatim}
2
\end{verbatim}

Next, combine the \texttt{rcvdEvt.a} fragment with the 2 fragment with the binary addition operator:

\begin{verbatim}
get_evt_data_item_a () + 2
\end{verbatim}

Finally, build an assignment statement from the binary operation:

\begin{verbatim}
x = get_evt_data_item_a () + 2;
\end{verbatim}

We extend this approach to all statement components in the Action Language.

A script can be generated to allow the user to furnish each fragment as it is needed - the script used for conversion of:

\begin{verbatim}
Assign x = rcvdEvt.a + 2;
\end{verbatim}

is made up of a series of calls to fragment generation functions (see Section 2.2.1.2) and would look like:

\begin{verbatim}
.Invoke a001 = actn_begin (actn_inst)
.Invoke b001 = blck_begin ()
.Invoke v001 = var_declare_self (actn_inst)
.Invoke r001 = rval_read_rcvd_evt_di (actn_inst, "a")
.Invoke r002 = rval_literal_integer ("2")
.Invoke r003 = rval_binary_op (r001, "+", r002)
.Invoke v002 = var_declare ("x")
.Invoke s001 = stmt_assign_transient_var (v002, TRUE, r003)
.Invoke b002 = blck_append_stmt (b001, s001)
\end{verbatim}
Notice that the script begins by requesting a fragment for the innermost statement component and passes that fragment into requests for larger fragments which eventually result in the whole statement converted to source code.

10.1.3 Fragment Generation Functions

A Fragment Generation Function exists for each Action Language Statement Component as well as the Action Language Statements, Statement Blocks, and the Action itself. The role of some Fragment Generation Functions is to build a fragment from scratch, i.e., from string parameters and Generation Database lookups. The role of other Fragment Generation Functions are to assemble small fragments into larger fragments.

Generator functions have been grouped into the following categories:

- Action
- Statement Block
- Statement
- Rvalue
- Instance Chain
- Parameter List

Each type of Generator Function will be discussed in the following sections.

10.1.3.1 Action Generator Functions

An action is really a block - however, special handling may be required, e.g., declaration statements may need to be output at the beginning of the action only. So, the Action Generator Functions provide the mechanism for this potential special handling.
10.1.3.2 Statement Block Generator Functions

These Generator Functions are similar to the Action Generator Functions only they operate on a block of Action Language Statements.

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>actn_begin</td>
<td>p_actn_obj_inst</td>
<td>inst_ref&lt;SM_ACT&gt;</td>
</tr>
<tr>
<td>actn_append_blck</td>
<td>p_actn</td>
<td>frag_ref&lt;actn&gt;</td>
</tr>
<tr>
<td></td>
<td>p_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
<tr>
<td>actn_end</td>
<td>p_actn</td>
<td>frag_ref&lt;actn&gt;</td>
</tr>
</tbody>
</table>

10.1.3.3 Statement Generator Functions

These Generator Functions correspond directly to the Action Language Statements - some Action Language Statements may have more than one Generator Function because of differing variations in that statement.

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>blck_begin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blck_append_stmt</td>
<td>p_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
<tr>
<td></td>
<td>p_stmt</td>
<td>frag_ref&lt;stmt&gt;</td>
</tr>
<tr>
<td>blck_var_out_of_scope</td>
<td>p_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
<tr>
<td></td>
<td>p_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td>blck_end</td>
<td>p_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
</tbody>
</table>
### TABLE 10.13 Statement Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt_select_related_by</td>
<td>p_cardinality</td>
<td>string (“ONE”, “ANY”, “MANY”)</td>
</tr>
<tr>
<td></td>
<td>p_select_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_is_implicit_decl</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>p_chain</td>
<td>frag_ref&lt;chain&gt;</td>
</tr>
<tr>
<td>stmt_select_from_instances_of</td>
<td>p_cardinality</td>
<td>string (“ANY”, “MANY”)</td>
</tr>
<tr>
<td></td>
<td>p_select_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_is_implicit_decl</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td>stmt_for</td>
<td>p_inst_ref_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_is_implicit_decl</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>p_inst_ref_set_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_for_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
<tr>
<td>stmt_create_obj_inst_no_var</td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td>stmt_create_obj_inst</td>
<td>p_inst_ref_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_is_implicit_decl</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td>stmt_delete_obj_inst</td>
<td>p_inst_ref_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td>stmt_relate</td>
<td>p_inst_ref_1_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_inst_ref_2_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_Rnum</td>
<td>integer</td>
</tr>
<tr>
<td>stmt_relate_using</td>
<td>p_inst_ref_1_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_inst_ref_2_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_Rnum</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>p_assoc_inst_ref_var</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>p_rel_phrase</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
</tbody>
</table>
### TABLE 10.13 Statement Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stmt_unrelate</code></td>
<td><code>p_inst_ref_1_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_inst_ref_2_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_Rnum</code></td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td><code>p_rel_phrase</code></td>
<td></td>
</tr>
<tr>
<td><code>stmt_unrelate_using</code></td>
<td><code>p_inst_ref_1_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_inst_ref_2_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_Rnum</code></td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td><code>p_assoc_inst_ref_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_rel_phrase</code></td>
<td></td>
</tr>
<tr>
<td><code>stmt_generate_obj_inst</code></td>
<td><code>p_evt_label</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_param</code></td>
<td>frag_ref&lt;param&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_inst_ref_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td><code>stmt_generate_assigner</code></td>
<td><code>p_evt_label_str</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_param</code></td>
<td>frag_ref&lt;param&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_obj_keyletters</code></td>
<td>string</td>
</tr>
<tr>
<td><code>stmt_generate.creation</code></td>
<td><code>p_evt_label</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_evt_frag</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>p_obj_kl</code></td>
<td></td>
</tr>
<tr>
<td><code>stmt_generate_ext_entity</code></td>
<td><code>p_evt_label</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_param</code></td>
<td>frag_ref&lt;param&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_ext_entity_keyletters</code></td>
<td>string</td>
</tr>
<tr>
<td><code>stmt_generate_ext_inst</code></td>
<td><code>p_evt_inst_var_frag</code></td>
<td></td>
</tr>
<tr>
<td><code>stmt_create_evt_obj_inst</code></td>
<td><code>p_evt_inst_var</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_is_implicit_decl</code></td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td><code>p_evt_label</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_param</code></td>
<td>frag_ref&lt;param&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_obj_inst_ref_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td><code>stmt_assign_obj_attr</code></td>
<td><code>p_inst_ref_var</code></td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td><code>p_attr_name</code></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><code>p_expression_rval</code></td>
<td>frag_ref&lt;rval&gt;</td>
</tr>
</tbody>
</table>
### TABLE 10.13 Statement Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt_assign_transient_var</td>
<td>p_transient_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_is_implicit_decl</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>p_expression_rval</td>
<td>frag_ref&lt;rval&gt;</td>
</tr>
<tr>
<td>stmt_transform_void</td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_method_name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_param</td>
<td>frag_ref&lt;param&gt;</td>
</tr>
<tr>
<td>stmt_if</td>
<td>p_condition_rval</td>
<td>frag_ref&lt;rval&gt;</td>
</tr>
<tr>
<td></td>
<td>p_if_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
<tr>
<td>stmt_else</td>
<td>p_else_blck</td>
<td>frag_ref&lt;blck&gt;</td>
</tr>
<tr>
<td>stmt_bridge_void</td>
<td>p_ext_entity_keyletters</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_method_name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_param</td>
<td>frag_ref&lt;param&gt;</td>
</tr>
</tbody>
</table>

### 10.1.3.4 Rvalue Generator Functions

These Generator Functions are to handle expressions which contain calculations and tests.

### TABLE 10.14 Rvalue Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rval_literal_boolean</td>
<td>p_boolean_value</td>
<td>string (&quot;TRUE&quot;, &quot;FALSE&quot;)</td>
</tr>
<tr>
<td>rval_literal_integer</td>
<td>p_integer_value</td>
<td>string</td>
</tr>
<tr>
<td>rval_literal_real</td>
<td>p_real_value</td>
<td>string</td>
</tr>
<tr>
<td>rval_literal_string</td>
<td>p_string_value</td>
<td>string</td>
</tr>
<tr>
<td>rval_read_rcvd_evt_di</td>
<td>p_actn_obj_inst</td>
<td>inst_ref&lt;SM_ACT&gt;</td>
</tr>
<tr>
<td></td>
<td>p_evt_di_name</td>
<td>string</td>
</tr>
<tr>
<td>rval_read_obj_attr</td>
<td>p_inst_ref_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
<tr>
<td></td>
<td>p_attr_name</td>
<td>string</td>
</tr>
<tr>
<td>rval_read_transient_var</td>
<td>p_transient_var</td>
<td>frag_ref&lt;var&gt;</td>
</tr>
</tbody>
</table>
10.1.3.5 Instance Chain Generator Functions

These Generator Functions support conversion of the instance chains in select statements to source code.

### TABLE 10.15 Instance Chain Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain_begin</td>
<td>p_cardinality</td>
<td>string (“ONE”, “ANY”, “MANY”)</td>
</tr>
<tr>
<td></td>
<td>p_begin_inst_ref_var</td>
<td>string</td>
</tr>
<tr>
<td>chain_add_link</td>
<td>p_chain</td>
<td>frag_ref&lt;chain&gt;</td>
</tr>
<tr>
<td></td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_Rnum</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>p_rel_phrase</td>
<td>string</td>
</tr>
<tr>
<td>chain_end</td>
<td>p_chain</td>
<td>frag_ref&lt;chain&gt;</td>
</tr>
</tbody>
</table>

10.1.3.6 Parameter List Generator Functions
Step 10: Develop Action Archetypes

### TABLE 10.16 Parameter List Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>param_begin_evt_obj_inst</td>
<td>p_evt_label</td>
<td>string</td>
</tr>
<tr>
<td>param_begin_evt_assigner</td>
<td>p_evt_label</td>
<td>string</td>
</tr>
<tr>
<td>param_begin_evt_creation</td>
<td>p_evt_label</td>
<td></td>
</tr>
<tr>
<td>param_begin_evt_ext_entity</td>
<td>p_evt_label</td>
<td>string</td>
</tr>
<tr>
<td>param_begin_transform</td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_method_name</td>
<td>string</td>
</tr>
<tr>
<td>param_begin_bridge</td>
<td>p_ext_entity_keyletters</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_method_name</td>
<td>string</td>
</tr>
<tr>
<td>param_add</td>
<td>p_param</td>
<td>frag_ref&lt;param&gt;</td>
</tr>
<tr>
<td></td>
<td>p_param_name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_param_rval</td>
<td>frag_ref&lt;rval&gt;</td>
</tr>
<tr>
<td>param_end</td>
<td>p_param</td>
<td>frag_ref&lt;param&gt;</td>
</tr>
</tbody>
</table>

### 10.1.3.7 Variable Generator Functions

### TABLE 10.17 Variable Generator Functions

<table>
<thead>
<tr>
<th>Generator Function Name</th>
<th>Parameter Name</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>vardeclare_self_obj_inst_ref</td>
<td>p_actn_obj_inst</td>
<td>inst_ref&lt;SM_ACT&gt;</td>
</tr>
<tr>
<td>vardeclare_obj_inst_ref</td>
<td>p_var_name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td>vardeclare_obj_inst_ref_set</td>
<td>p_var_name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td>vardeclare_evt_inst</td>
<td>p_var_name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>p_obj_keyletters</td>
<td>string</td>
</tr>
<tr>
<td>vardeclare</td>
<td>p_var_name</td>
<td>string</td>
</tr>
</tbody>
</table>
10.2 Automation

10.2.1 Overview
Action Source Code is generated using a technique very similar to that used by a compiler to produce machine code from a high level language. However, one major difference exists: the last step of the Action Generation is done through a set of user-specified Fragment Generation Functions so that the user has complete control over the process of generation and therefore can generate source code in any Programming Language (C, C++, Smalltalk, Fortran, COBOL, Lisp, Assembler, 4GL... and any Software Architecture.

Figure 10.2.1.1. Archetype Interpretation Architecture
Figure 10.2.1.2. Action Source Code Generation.

Script Generator

Action Specification

Select many cats from instances of CAT; For each cat in cats Generate DG1 to cat; End for;

Fragment Generation Script

Select many cats from instances of CAT; For each cat in cats Generate DG1 to cat; End for;

Action Source Code Generator

Select many cats from instances of CAT; For each cat in cats Generate DG1 to cat; End for;

Fragment Generator Functions

Select many cats from instances of CAT; For each cat in cats Generate DG1 to cat; End for;

Source Code

Figure 10.2.1.2 shows the steps involved in Action Generation.
Figure 10.2.1.3 shows the Action Generation Process.

```java
for (int i = 0; i < cardinality; i++) {
    v1 = var_declare_inst_ref(v1, “CAT”);
    b1 = blck_append_stmt(b1, a1 = actn_append_blk);
}
```

Action Language
(user specified)

Select many cats from instances of CAT;
For each cat in cats
    Generate DG1 to cat;
End for;

Source Code
Final Product

Action Language
(user specified)

Figure 10.2.1.3. Action Generation Process
Step 10: Develop Action Archetypes
Index

Symbols
  148
- 148
!= 148
% 148
* 148
+ 148
/ 148
// 142
== 148
> 148
>= 148

A
Access Path 88
Action 110
Action Generator Functions 166
Action Home 110
Action Source Code Generation 162
actn_append_blk 167
actn_begin 167
actn_end 167
AL_xlate 163
and 148
any 128
Architectural Decisions 8
Architecture Blueprint 1
Architecture Characterization 3
Architecture Design 13
Assign 135
Assign Statement 135
Assigner State Model 113
assigner_sm 163
Associative Relationship 69
Attribute 46
Attribute Reference in Object 51
Automation 155, 173

B
Base Attribute 48
blck_append_stmt 167
blck_begin 167
blck_end 167
blck_var_out_of_scope 167
boolean 136
Bridge 31
Bridge Parameter 32
BridgePoint - Automation iii
BridgePoint - OOA iii
BridgePoint - Tool Guide iii

C
CA_ACC 88
CA_COMM 84
CA_EESMC 84
CA_EESME 86
CA_SMEEA 89
CA_SMEEC 85
CA_SMEEE 90
CA_SMEEE 87
CA_SMOA 88
CA_SMOA AA 89
CA_SMSMC 85
CA_SMSME 87
Cant Happen 105
cardinality 147
chain_add_link 171
chain_begin 171
chain_end 171
Clear 142
Comment 142
Comments 142
Communication Path 84
Composition Relationship 73
Compound Expressions 146
Control Organization 10
Index

Core Data Type 30
Creation Transition 107

D
Data Access Control Statements 128
Data Organization 8
Data Type 29
Database 7
Derived Base Attribute 49
Develop Action Archetypes 159
Develop Structural Archetypes 123
Documentation Roadmap iii
Domain 20

E
EE to SM Comm Path 84
EE to SM Event Comm 86
elif 137
else 137
Emit to file 142
Emitting Generated Output 141
empty 147
end function 139
end if 137
Event Ignored 105
Event Supplemental Data 112
Exit 143
Expressions 145
External Entity 22
External Entity Data Item 24
External Entity Event 25
External Entity Event Data 28
External Entity Event Data Item 27
External Entity in Model 23

F
File Control Statements 141
first 147
Format 152
frag_ref 136
Fragment Attributes 139
Fragment Generation 163
Fragment Generation Functions 166
Fragment Generation Script 164
from instances of 128
Function 138
Function Control Statements 138

G
gen_file 156
gen_import 156
General Language Attributes 126

H
Handling Errors 143

I
If 137
Implementation Technologies 6
Implemented Service Domains 8
Imported Object 45
Include 143, 163
info.arch_file_line 153
info.arch_file_name 153
info.date 153
info.interpreter_platform 154
info.interpreter_version 154
info.unique_num 154
info.user_id 153
Information Substitution Variables 153
inst_ref 136
inst_ref_set 136
Instance Chain Generator Functions 171
Instance Selection 128
Instance State Model 112
instance_sm 163
integer 136
Invoke 139

L
last 147
Literal Text 127
Literals as Rvalues 144

M
many 128
Mealy Action Home 109
Mealy State Model 108
Method 5, 15, 125, 161
Moore Action Home 109
Moore State Model 108

N
New Base Attribute 49
New State Transition 104
No Event Transition 107
not 147
not_empty 147
not_first 147
not_last 147
Index

O
O_ATTR 46
O_BATTR 48
O_DBATTR 49
O_ID 47
O_IOBJ 45
O_NBATTR 49
O_OBJ 44
O_OID 48
O_RATTR 50
O_REF 51
O_RTIDA 53
O_TFR 53
O_TPARM 54
Object 44
Object and Attribute Descriptions 20, 44, 64, 84, 100
Object As Associated One Side 69
Object As Associated Other Side 70
Object As Associator 71
Object As Composition One Side 74
Object As Composition Other Side 75
Object As Simple Formalizer 68
Object As Simple Participant 67
Object As Subtype 73
Object As Supertype 72
Object Attribute Access 135
Object Identifier 47
Object Identifier Attribute 48
Object in Relationship 64
one 128
Operating System 6
Operations 147
or 148

P
Param 138
param_add 172
param_begin_bridge 172
param_begin_evt_assigner 172
param_begin_evt_ext_entity 172
param_begin_evt_obj_inst 172
param_begin_transform 172
param_end 172
Parameter List Generator Functions 171
Parse Keyword 153
Print 143
Programming Language 6

Q
Quoted Strings 144
R
R_AONE 69
R_AOTH 70
R_ASSOC 69
R_ASSR 71
R_COMP 73
R_CONE 74
R_COTH 75
R_FORM 68
R_OIR 64
R_PART 67
R_REL 64
R_RGO 66
R_RTO 65
R_SIMP 66
R_SUB 73
R_SUBSUP 72
R_SUPER 72
real 136
Referential Attribute 50
Referred To Identifier Attribute 53
Referred To Object in Rel 65
Referring Object in Rel 66
related by 128
Relationship 64
Relationship Descriptions 34, 56, 77, 91, 114
rval_binary_op 171
rval_bridge 171
rval_literal_boolean 170
rval_literal_integer 170
rval_literal_real 170
rval_literal_string 170
rval_read_obj_attr 170
rval_read_rcvd_evt_di 170
rval_read_transient_var 170
rval_transform 171
rval_unary_op 171
Rvalue Generator Functions 170
Rvalues 143
S
S_BPARM 32
S_BRG 31
S_CDT 30
S_DOM 20
S_EE 22
S_EEDI 24
S_EEEDI 27
S_EEEDT 28
S_EEEVT 25
S_EEM 23
S_SS 21
S_UDT 30
Script Generation 162
script to file 163
Select 128
Simple Expressions 145
Simple Relationship 66
SM to EE Access Path 89
SM to EE Comm Path 85
SM to EE Data Item Access 90
SM to EE Event Comm 87
SM to OBJ Access Path 88
SM to OBJ Attribute Access 89
SM to SM Event Comm 87
SM_ACT 110
SM_AH 110
SM_ASM 113
SM_CH 105
SM_CRTXN 107
SM_EIGN 105
SM_EVT 101
SM_EVTDI 111
SM_ISM 112
SM_MEAH 109
SM_MEALY 108
SM_MOAH 109
SM_MOORE 108
SM_NETXN 107
SM_NSTXN 104
SM_SDII 112
SM_SEME 103
SM_SM 100
SM_STATE 100
SM_SUPDT 112
SM_TXN 106
Source Code Organization 11
Special Characters in Literal Text Lines 127
State Event Matrix Entry 103
State Model 100
State Model Event 101
State Model Event Data Item 111
State Model State 100
Statement Block Generator Functions 167
Statement Generator Functions 167
stmt_assign_obj_attr 169
stmt_assign_transient_var 170
stmt_bridge_void 170
stmt_create_evt_obj_inst 169
stmt_create_obj_inst 168
stmt_create_obj_inst_no_var 168
stmt_delete_obj_inst 168
stmt_else 170
stmt_for 168
stmt_generate_assigner 169
stmt_generate_ext_entity 169
stmt_generate_obj_inst 169
stmt_if 170
stmt_relate 168
stmt_relate_using 168
stmt_select_from_instances_of 168
stmt_select_related_by 168
stmt_transform_void 170
stmt_unrelate 169
stmt_unrelate_using 169
string 136
Substitution Variables 151
Subsystem 21
Subsystem ‘Communication & Access’ 81
Subsystem ‘Object’ 41
Subsystem ‘Relationship’ 61
Subsystem ‘State Model’ 97
Subtype/Supertype Relationship 72
Supplemental Data Items 112
SW Arch Implementation 121
Syntax Notation 126

T
Tester Control Statements 137
to SM Comm Path 85
Transformer 53
Transformer Control Statements 135
Transformer Parameter 54
Transition 106

U
User Data Type 30
User Interface 7
Using Makefiles 157

V
var_declare 172
var_declare_evt_inst 172
var_declare_obj_inst_ref 172
var_declare_obj_inst_ref_set 172
var_declare_self_obj_inst_ref 172
Variable Generator Functions 172
Variables as Rvalues 145
<table>
<thead>
<tr>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>where 128</td>
</tr>
<tr>
<td>While 133</td>
</tr>
</tbody>
</table>