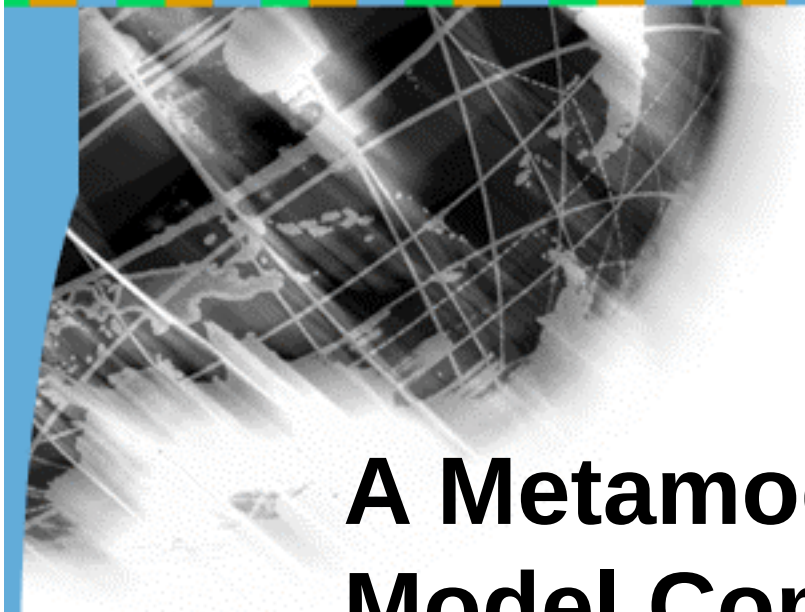


A Metamodel-Based Model Compiler

Shlaer-Mellor User Group Conference

May 23-24, 2000

Chuck Harvey

Dan Walden

Cambridge Technology Partners

BUILDING THE NEW ECONOMY

Overview / Topics



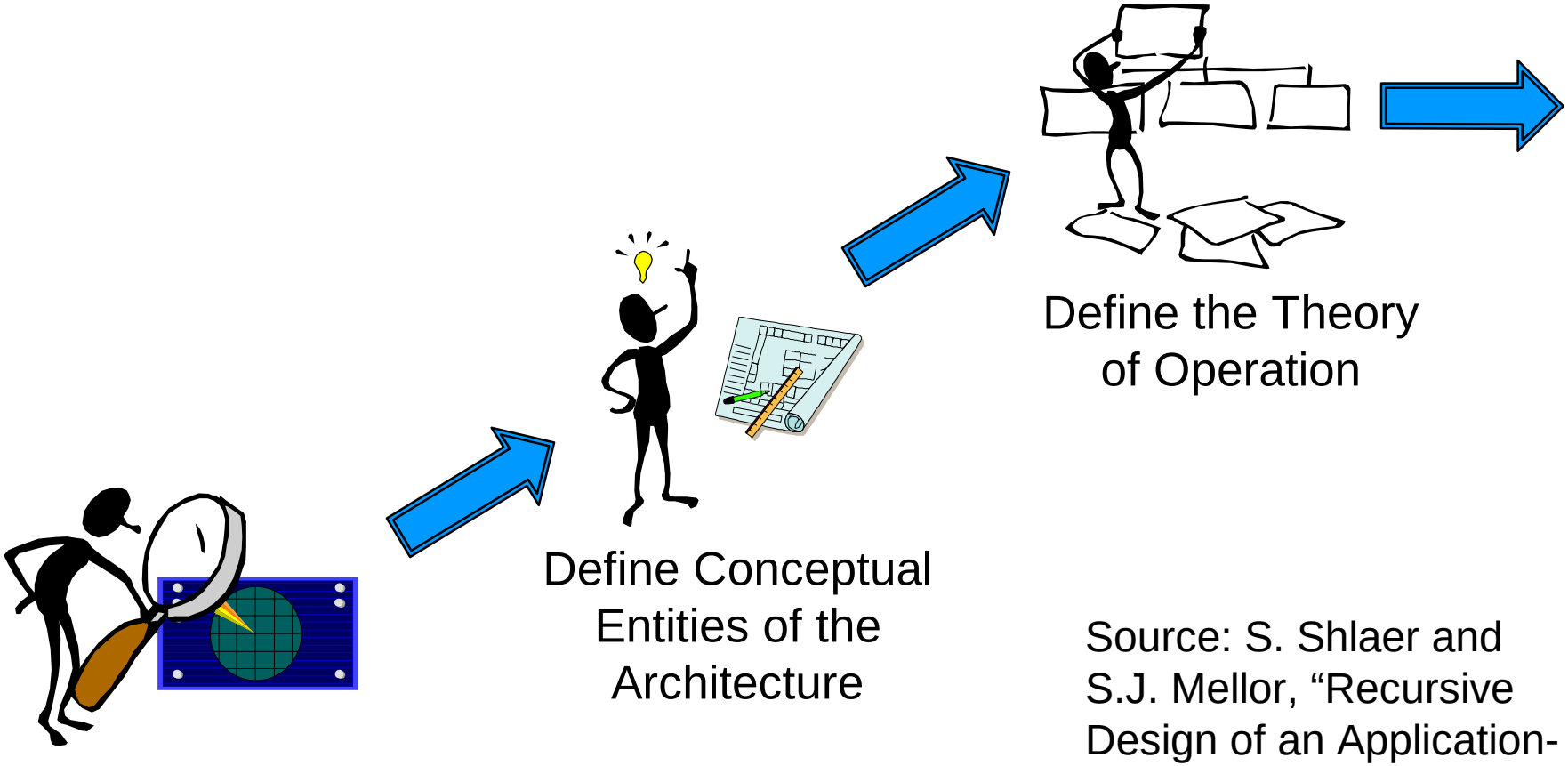
- Background
- Recursive design process and model compiler's role
- Metamodel-based model compiler design
  - Requirements
  - Structure
  - Control
- Future enhancements
- Conclusions and observations

Background



- Why develop a model compiler?
  - Off-the-shelf model compiler not available
  - CORBA / Java implementation environment
- Why develop a metamodel-based model compiler?
  - Approach supports “natural partitioning of work,” understandable
  - Reuse on future projects
  - An eye toward a family of compilers

Recursive Design Process



Define the Theory of Operation

Define Conceptual Entities of the Architecture

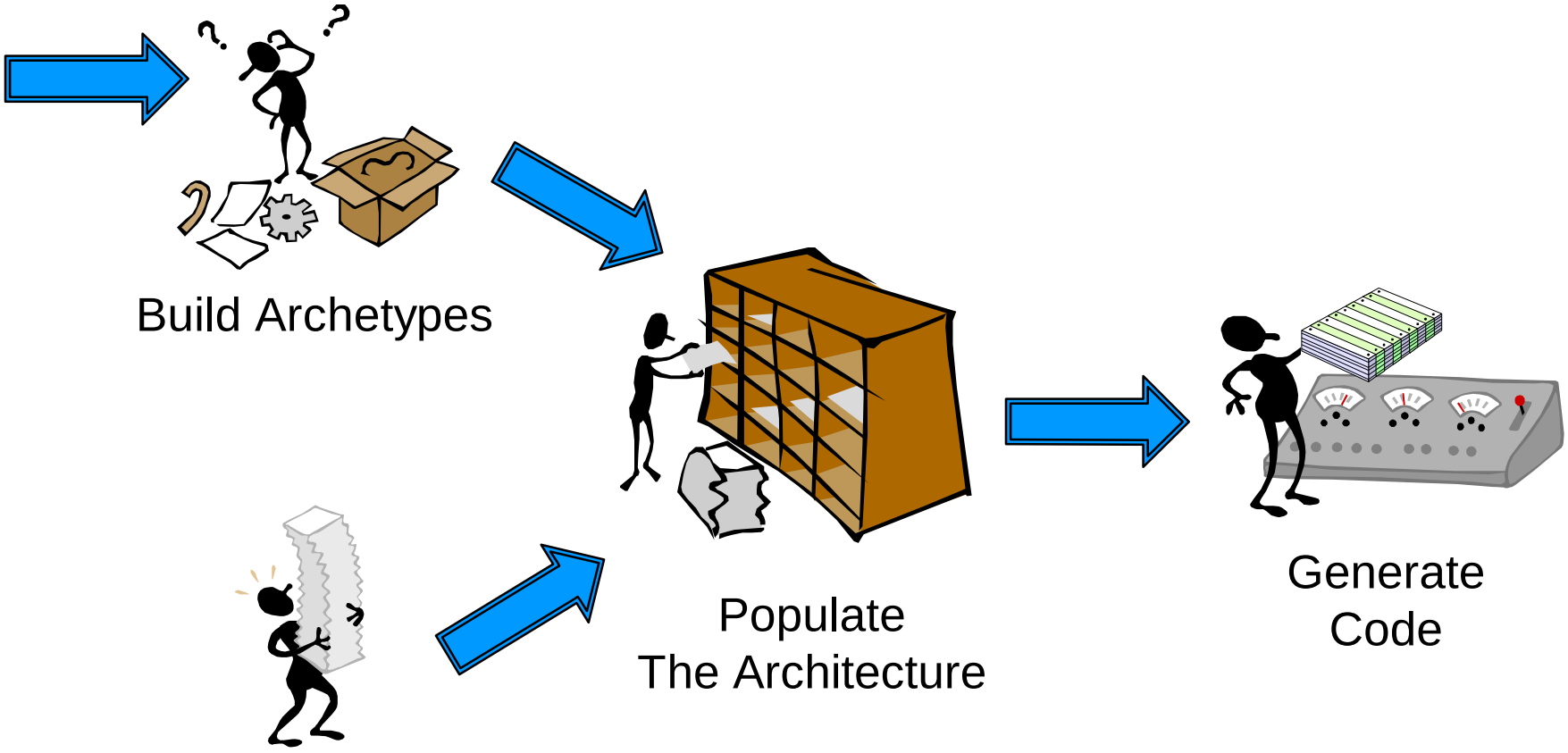
Characterize the System

Source: S. Shlaer and S.J. Mellor, “Recursive Design of an Application-Independent Architecture,” *IEEE Software*, Jan. 1997, pp. 61-72.

Cambridge Technology Partners

BUILDING THE NEW ECONOMY

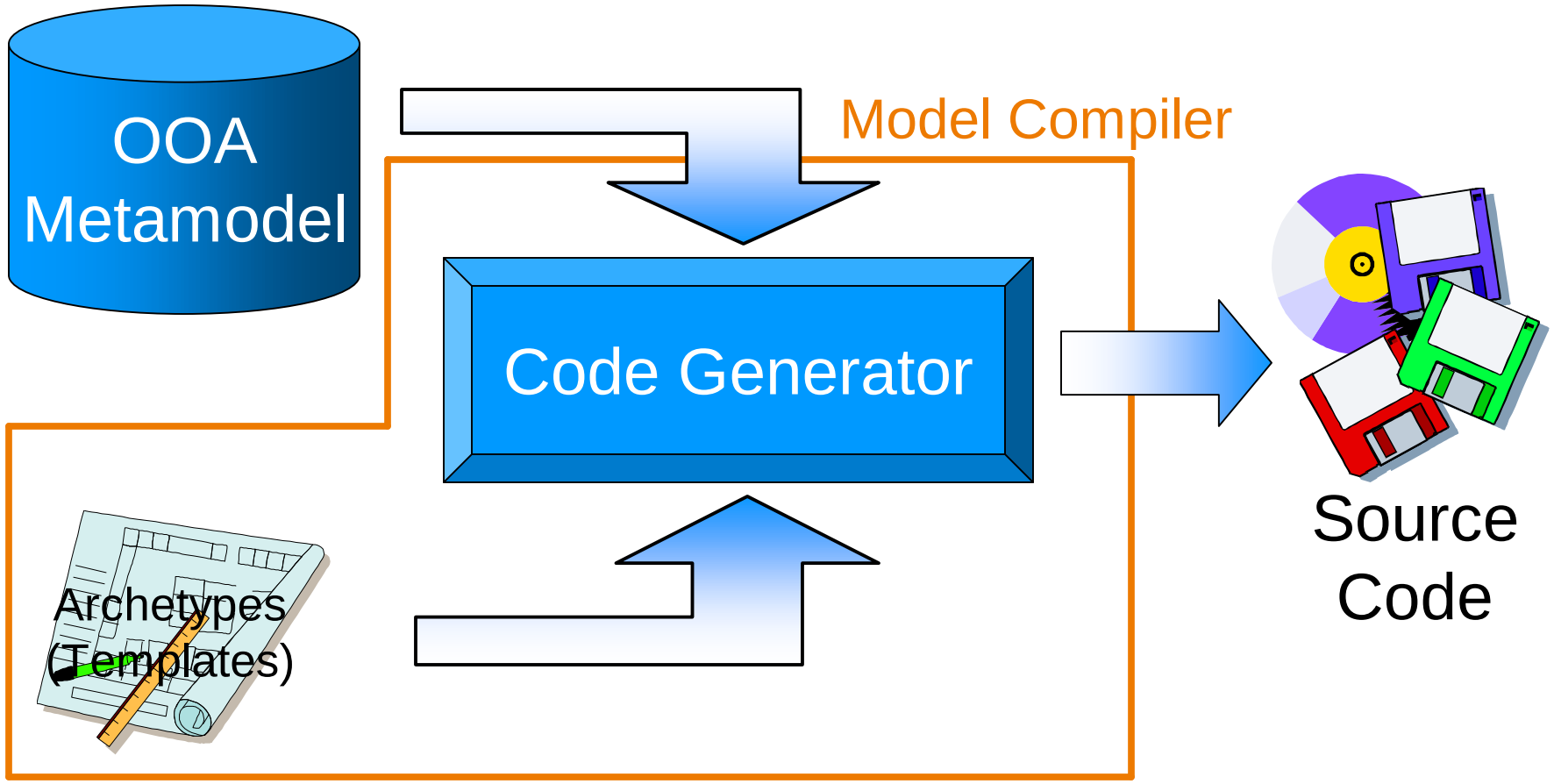
Recursive Design Process - Continued



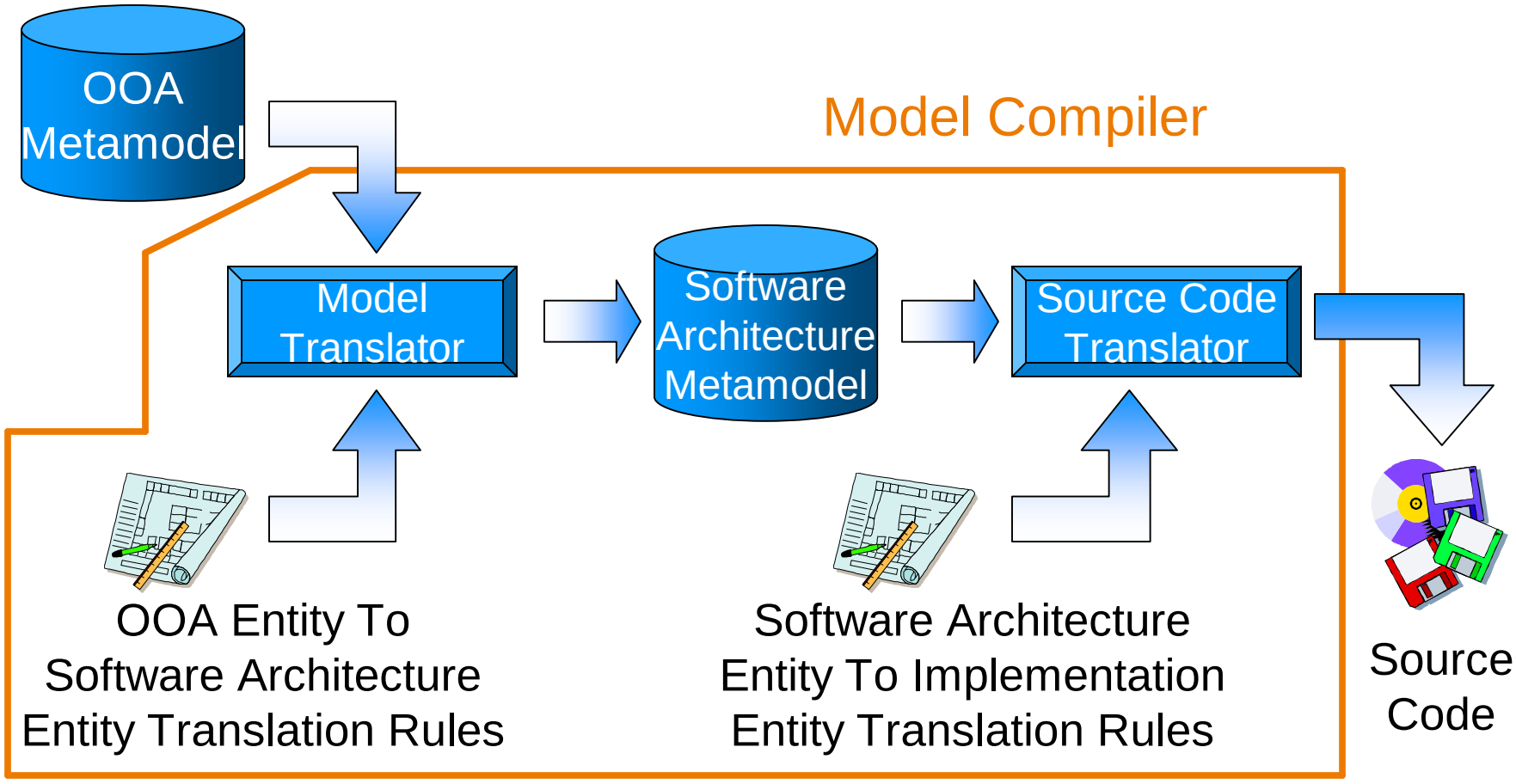
Model Compiler's Role

RD Process	Model Compiler Impact
Characterize the system	-
Define conceptual entities	Implemenation form (code or data)
Define theory of operation	-
Collect instance data	OOA mapping, coloring, allocation
Populate the architecture	Architecture metamodel population
Build the archetypes	Archetype structure
Generate code	Archetype processing

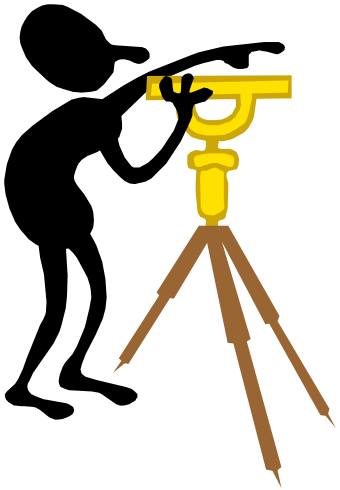
Code Generation Through Translation - Concept



Code Generation Through Translation - Metamodel Based

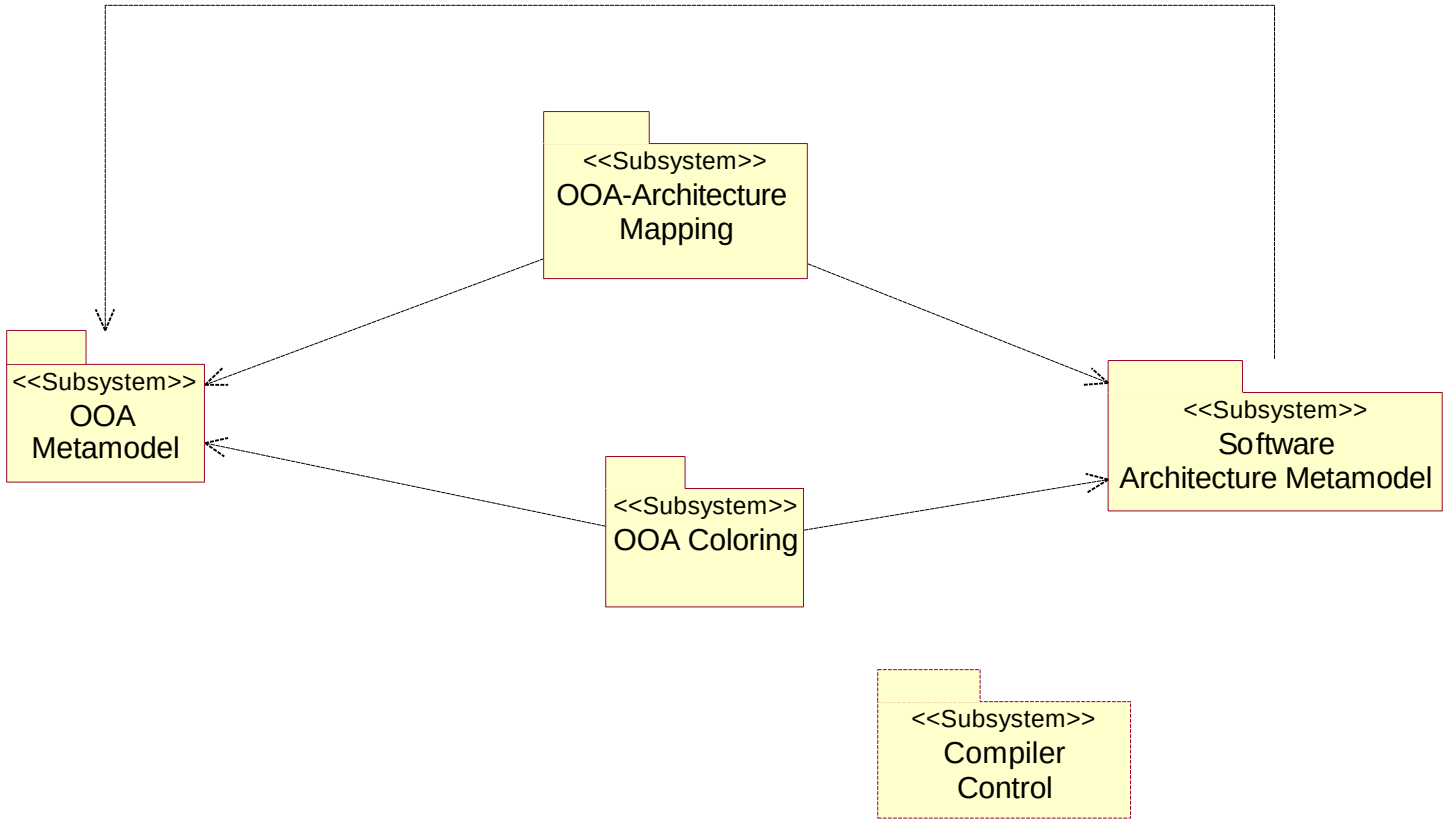


Model Compiler Requirements and Implementation Environment



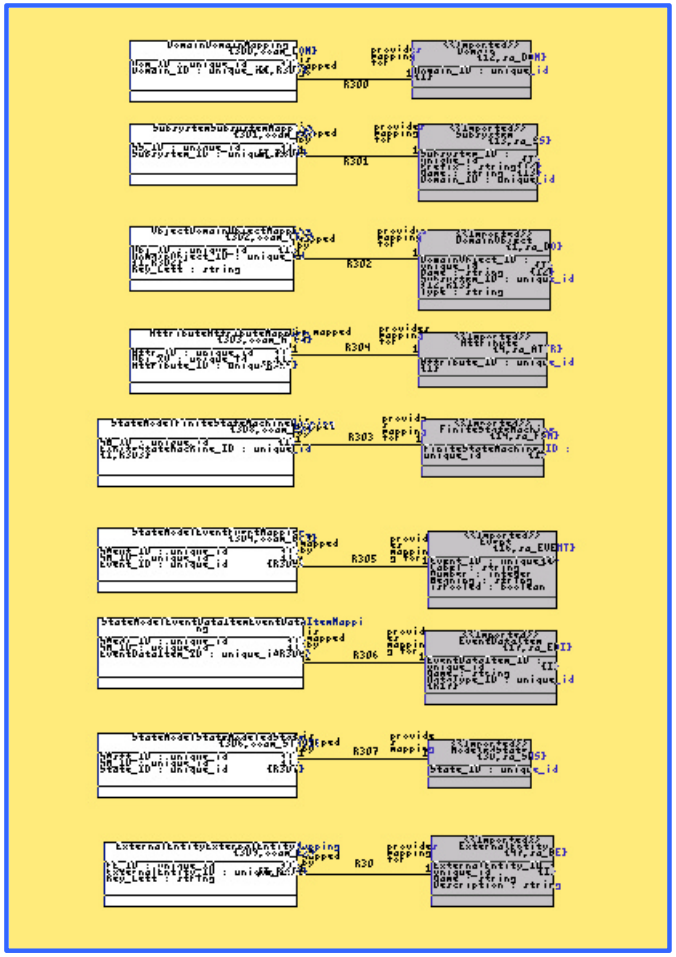
- Project tool, not a commercial compiler
- Two target languages: CORBA Interface Definition Language (IDL) and Java
- OOA Models captured using BridgePoint visual modeling tool (Version 4.0)
  - OOA metamodel
  - Archetype language
- Project Technology DesignPoint TinyJava BPAL components

Model Compiler Subsystem Relationship Diagram



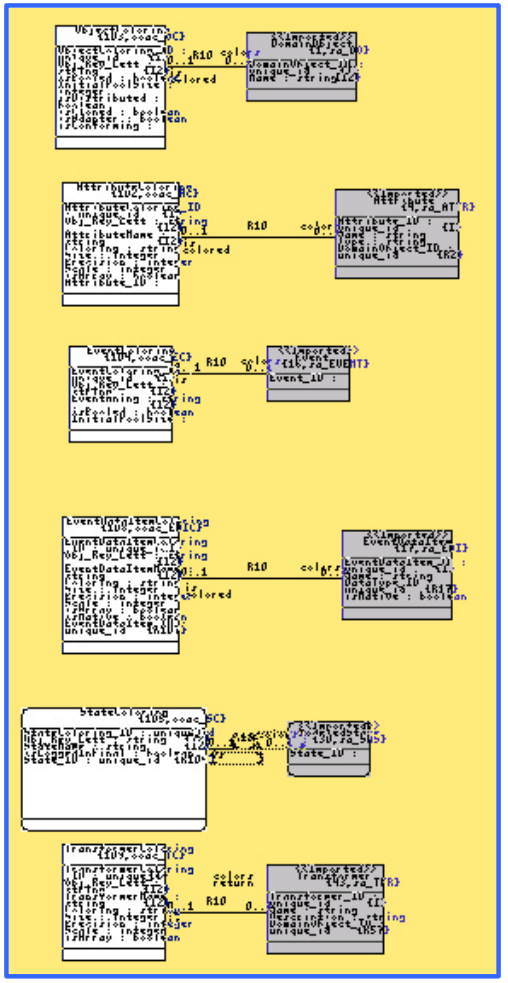
OOA-Architecture Mapping Subsystem

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



OOA Coloring Subsystem

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

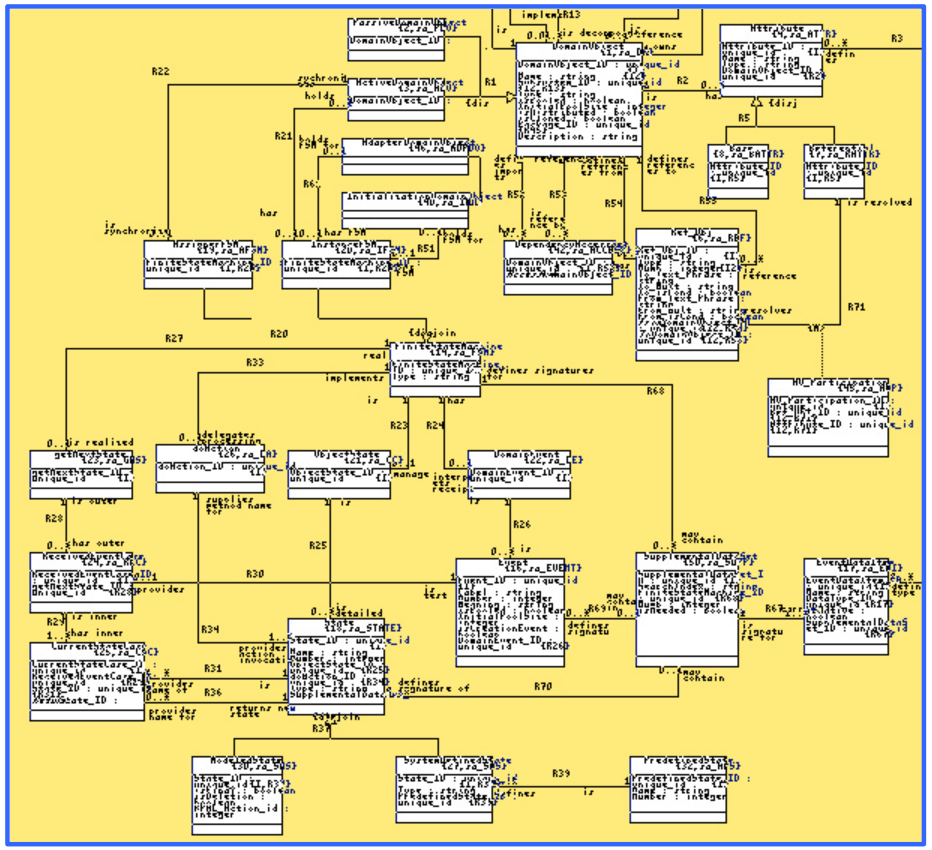


Cambridge Technology Partners

BUILDING THE NEW ECONOMY

Software Architecture Subsystem

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



Cambridge Technology Partners

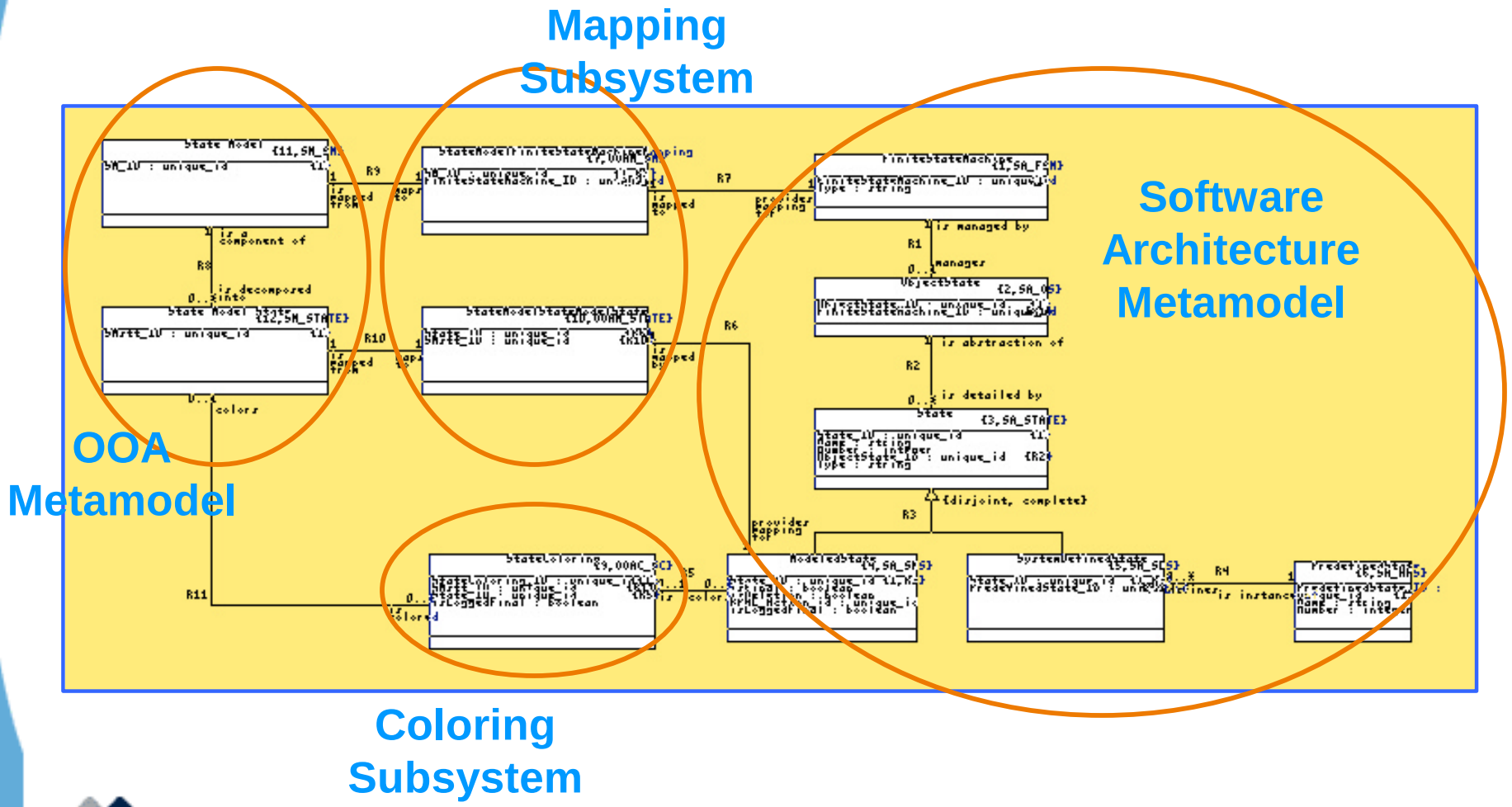
BUILDING THE NEW ECONOMY

Model Compilation Process

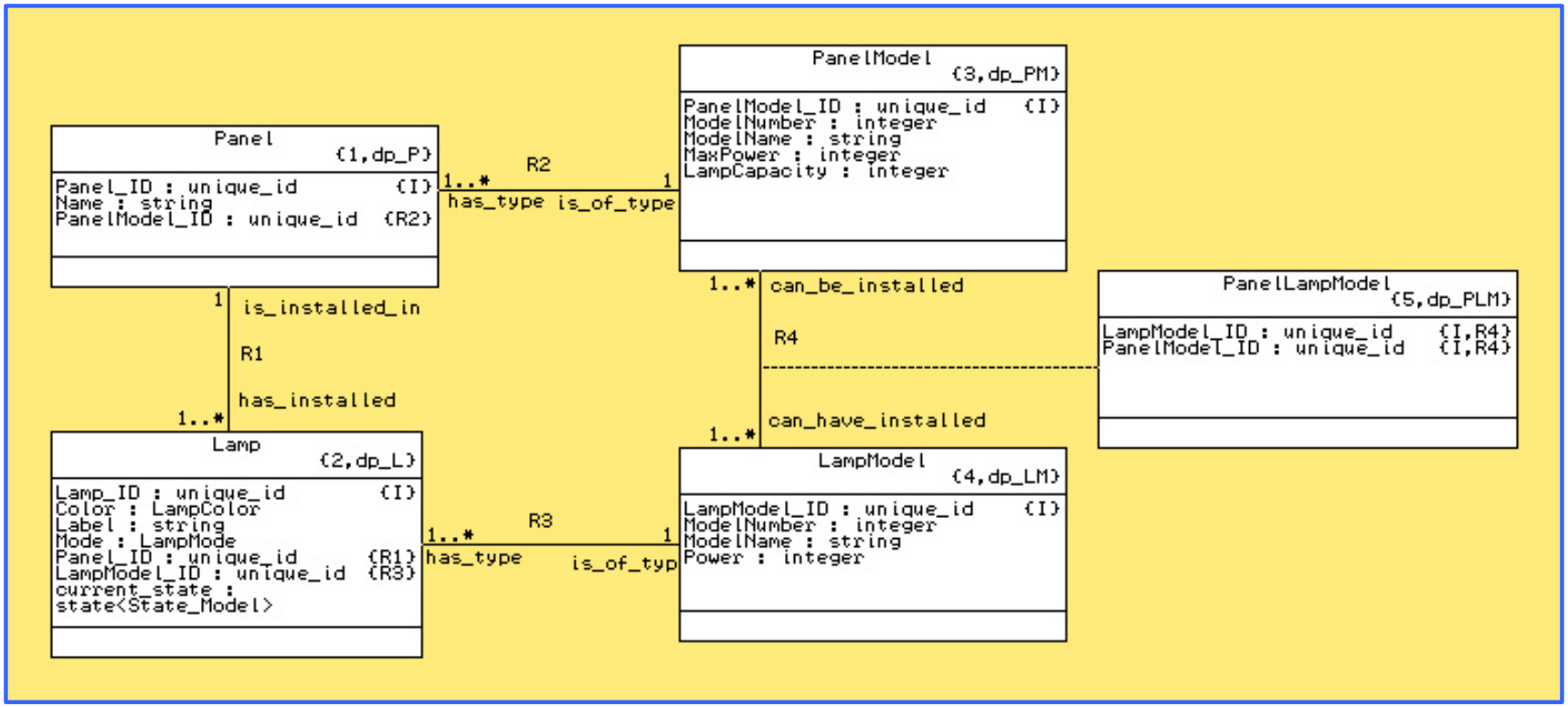


- Initialization
- Population of OOA metamodel, including BPAL
- Population of software architecture metamodel
  - Software architecture subsystem data types
  - Mapping and coloring subsystems
  - Remaining architecture subsystem entities
- Production of source code
  - Java, including translation of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

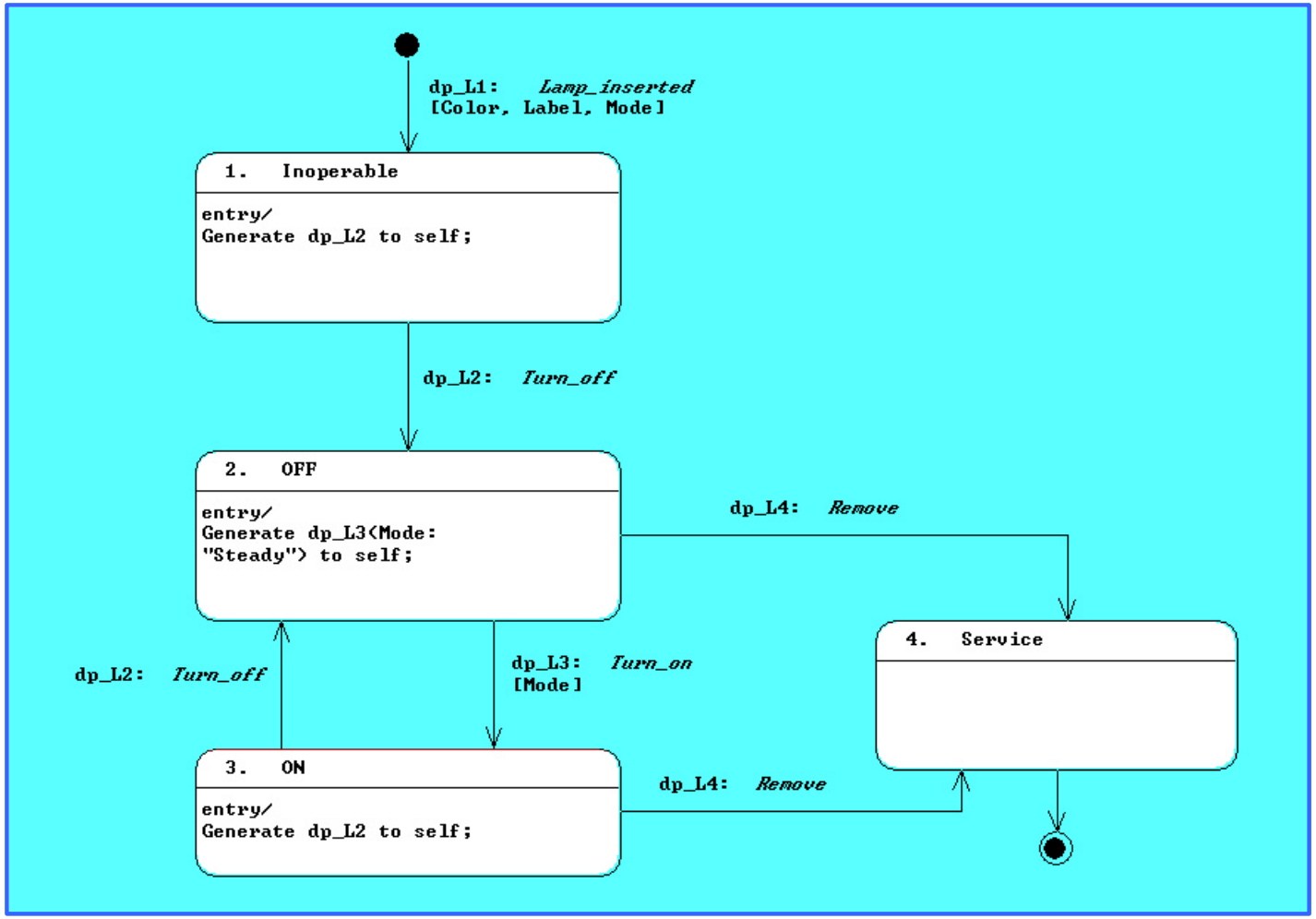
State Model State Subset



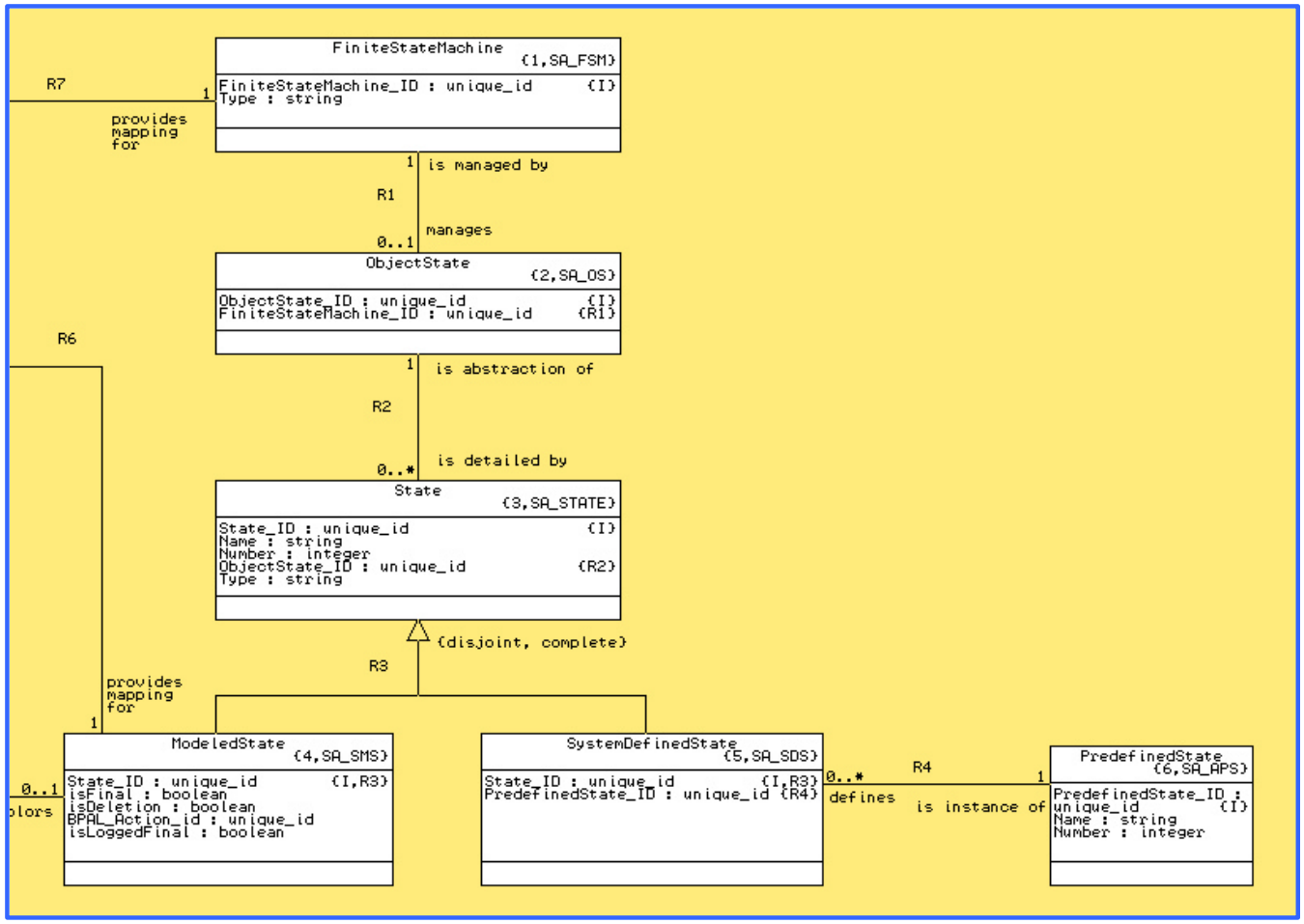
Example Object Information Model



Example State Model



Preexisting Instance Population



Populated Example Preexisting Instance Object

State

State_ID	*Name*	*Number*	*Type*
98	_UNDEFINED	0	SystemDefinedState
99	_DONOTCARE	-1	SystemDefinedState
100	_CANNOTHAPPEN	-2	SystemDefinedState
101	Inoperable	1	ModeledState
102	OFF	2	ModeledState
103	ON	3	ModeledState
104	Service	4	ModeledState

PredefinedState

Predefined State_ID	*Name*	*Number*
1	_UNDEFINED	0
2	_DONOTCARE	-1
3	_CANNOTHAPPEN	-2

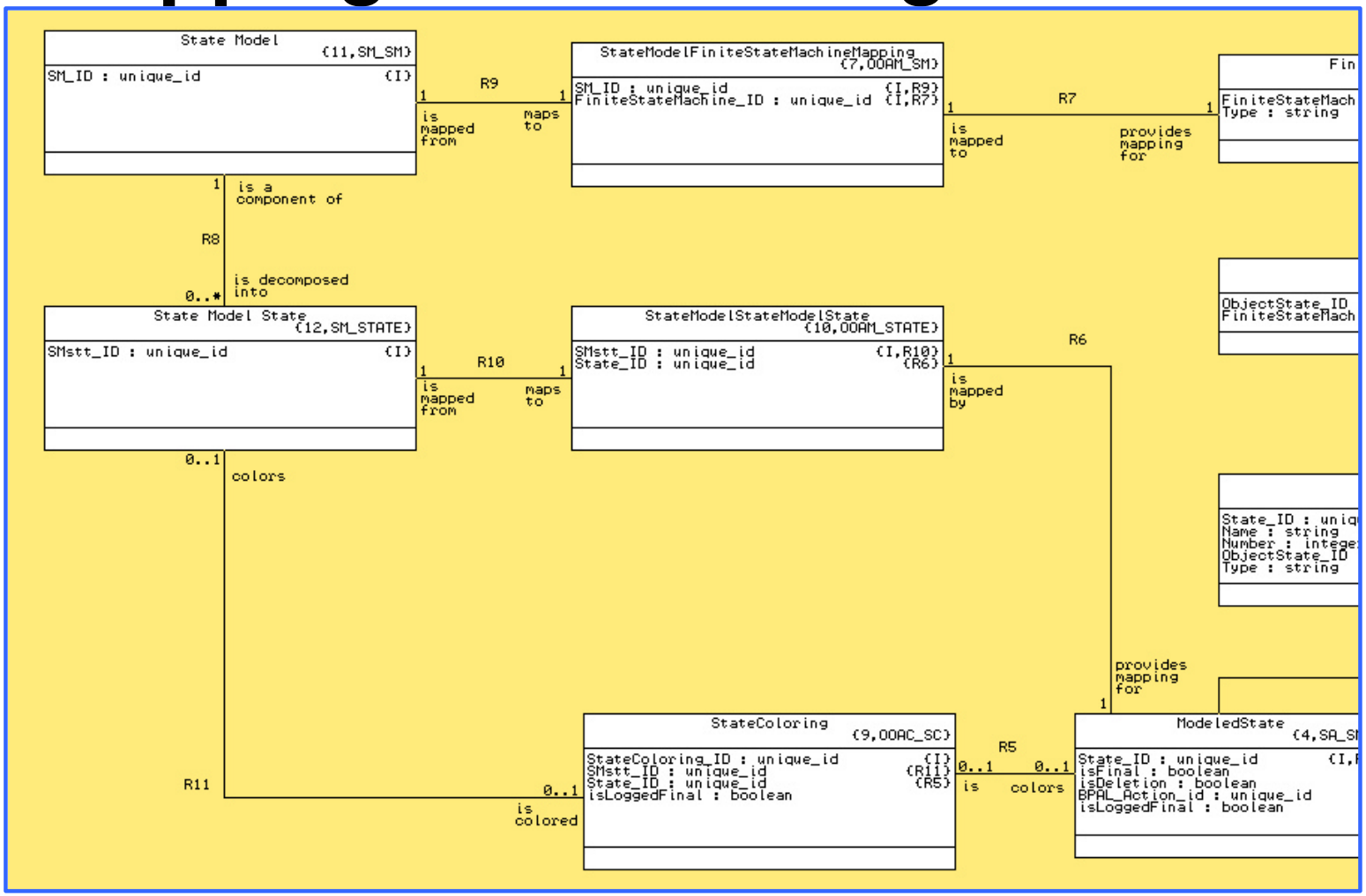
ModeledState

State_ID	*isLoggedInFinal*
101	False
102	False
103	False
104	True

SystemDefinedState

State_ID	*Predefined State_ID*
98	1
99	2
100	3

Mapping And Coloring



Cambridge Technology Partners

BUILDING THE NEW ECONOMY

Populated Example Mapping and Coloring Objects

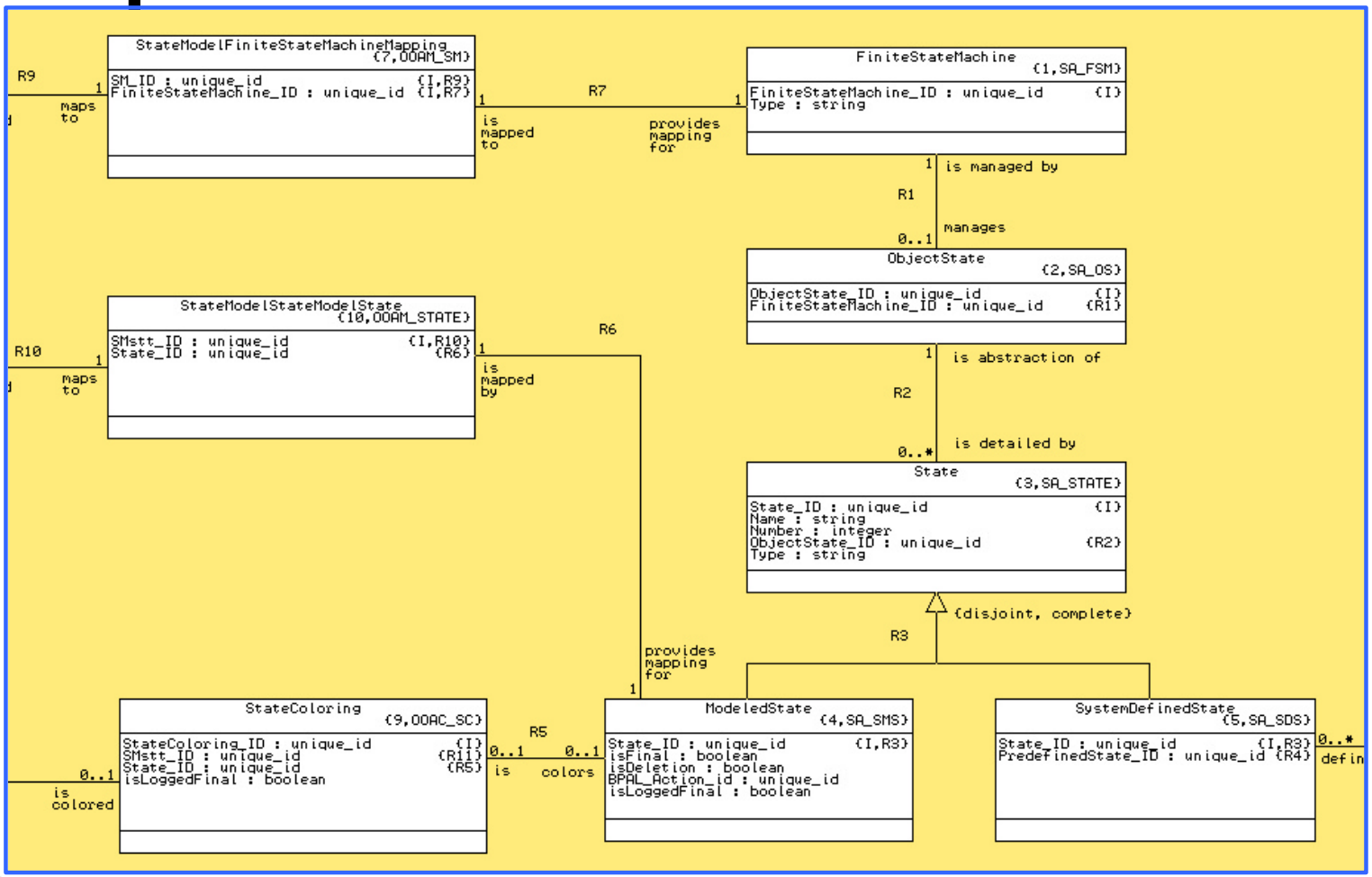
State Model State			
SMstt_ID	***Name***	***Numb***	***Final***
1001	Inoperable	1	
1002	OFF	2	
1003	ON	3	
1004	Service	4	1

StateModelStateModelState	
SMstt_ID	***State_ID***
1001	101
1002	102
1003	103
1004	104

StateColoring		
SMstt_ID	***isLoggedInFinal***	***State_ID***
1004	True	104

ModeledState	
State_ID	***isLoggedInFinal***
101	False
102	False
103	False
104	True

Software Architecture Metamodel Population



Cambridge Technology Partners

BUILDING THE NEW ECONOMY

Populated Example Software Architecture Objects

State			
State_ID	***Name***	***Number***	***Type***
98	_UNDEFINED	0	SystemDefinedState
99	_DONOTCARE	-1	SystemDefinedState
100	_CANNOTHAPPEN	-2	SystemDefinedState
101	Inoperable	1	ModeledState
102	OFF	2	ModeledState
103	ON	3	ModeledState
104	Service	4	ModeledState

PredefinedState		
Predefined State_ID	***Name***	***Number***
1	_UNDEFINED	0
2	_DONOTCARE	-1
3	_CANNOTHAPPEN	-2

ModeledState	
State_ID	***isLoggedInFinal***
101	False
102	False
103	False
104	True

SystemDefinedState	
State_ID	***Predefined State_ID***
98	1
99	2
100	3

Populator Archetype Design

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

Generated Example Java Source

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

Future Enhancements



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

Conclusions And Observations



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

Thank You!

Questions,
Anyone

