Building Bridges
A Preliminary Report

Sally Shlaer

Project Technology, Inc.
2560 Ninth Street, Suite 214
Berkeley, California 94710

29 August 1994

The purpose of this paper is to describe precisely what you need to do to define a bridge. Please note that this is a preliminary report: What is contained herein is believed to be correct, but not necessarily complete -- there may be cases that have not been covered here explicitly. To facilitate continuation of this work, I invite the reader to submit any examples that he or she has encountered and that have not been treated here.

This paper is organized into five major sections. In the first section we list the assumptions we make throughout the paper. The next three sections examine bridges from the perspectives of data, algorithm, and control. The final section provides a short discussion on bridges that mix elements of data, algorithm and control.

1. Assumptions

We make the following assumptions throughout this paper:

1. Both domains that participate in a bridge to be built have been completely and correctly analyzed in OOA.

2. The dependency between the domains is unidirectional[1] -- on each bridge on the domain chart, one domain plays the role of client and client only, while the other that of the server.

3. We assume that, prior to building a bridge, the data describing pre-existing

---

1While mutually dependent domains can, in principle, exist, we find this case to be quite rare. The technique described here has worked on those few mutually dependent domains we have encountered; however, this subject needs to be considered further to determine whether or not this is a fundamental observation or simply ongoing good luck.

instances has been collected for each of the participating domains[1]. This data should be placed in a separate database for each domain. Such an "instance database" is, of course, formed in accordance with the IM for this domain.

4. A bridge is conceptually similar to a clear pane of glass: It forms a barrier but contains nothing in the barrier. The bridge is only a correspondence (mapping) between items in one domain and items -- with different names and/or forms -- in the other domain.

5. Only enumerated mappings will be employed[2]. As a consequence, all mappings between domains can be expressed in tables.

6. It is the responsibility of the server domain to supply the form of the tables ("bridge tables") that are used to specify the bridge. These tables may be entirely empty or they may be partially populated with elements from the server domain. In either case, the server must supply instructions that allow the client to populate the bridge tables with elements from the client domain. Finally, the bridge tables are returned to the server to complete the population if necessary.

2. Data

2.1 Data Mappings

The issue in defining the data aspects of a bridge is to make a correspondence, or mapping, between data elements (objects, attributes, attribute values, instances, etc.) of the client domain and data elements of the server. Mappings may be and typically are made between different kinds of elements. For example:

> For all domains that are clients of the Architecture 4" domain, make an *instance* of the Class object in the Architecture for each *object* on a client's IM.

2.2 Bridge Tables for Data

2.2.1 Forms for Single Attribute Identifiers

Every bridge table is made up of two halves: one specifying the elements in the server, and the other specifying corresponding elements in the client. To make the appropriate empty bridge table for a data mapping involving an object whose identifier consists of a

---

1This work can begin as soon as a rudimentary IM has been developed. In practice, we find it beneficial to begin data collection quite early, because dealing with real data often helps to focus the analysis effort; in addition, it can help the analyst to form correct abstractions.
2This is not a restriction, since any mapping that can be expressed as a function can also be expressed as a table.

single attribute, select two of the following standard forms -- one for the server and one for the client. Note that these "half tables" have been drawn from an OOA of OOA.

Object (sai)[1]

Object name	Identifying attribute name

Attribute

Object name	Attribute name

Domain of Attribute (enumerated)

Object name	Attribute name	Attribute value

Domain of Attribute (range)

Object name	Attribute name	Low value	High value

Identifier (sai)

Object name	Identifying

1sai indicates that the object has a single attribute identifier.

	attribute name

Instance of Object (sai)

Object name	Identifying Attribute name	Identifying Attribute value

Attribute for instance (sai)

Object name	Identifying attribute name	Identifying attribute value	Attribute name

Attribute value (enumerated) for instance

Object name	Identifying attribute name	Identifying attribute value	Attribute name	Attribute value

Attribute value (range) for instance

Object name	Identifying attribute name	Identifying attribute value	Attribute name	Low value	High value

Additional columns may be added to these half tables provided that the information they convey is functionally dependent on the columns specified as primary key (underlined) in the standard forms. Columns that are not part of the primary key may be omitted if they are not otherwise needed to carry bridge information.

Once the two required half-tables have been developed, they are then glued together in the following manner to make the required complete empty bridge table:

Client			***Server***		
(label)	(label)	. . .	(label)	(label)	. . .

The server domain must then provide instructions that allow the client to populate all or part of the table. If these instructions would cause the creation of

- a column in which all cells contain the same entry
- a column whose entries exactly duplicate those of another column
- a column whose entries can be determined from another previously specified bridge table

this column may be eliminated from the table before handing it to the client domain for population. Such a column will be shown as shaded in the remainder of this paper.

The construction of bridge tables for data will be demonstrated in section 2.3 by several examples.

2.2.2 Forms for Multiple Attribute Identifiers

The tables in section 2.2.1 were formulated under the assumption that each object has an identifier consisting of a single attribute. Special forms are required when an object has an identifier made up of multiple attributes. The first such form has the effect of temporarily supplying an arbitrary single-attribute identifier only for the purpose of building a bridge:

Identifier (mai)[1]

Object name	Arbitrary identifying value	Identifying Attribute name	Identifying Attribute value

An entry must be made in the Identifier (mai) table for each attribute comprising the identifier of the object.

The Identifier form may be used in conjunction with tables employing any of the following multiple-attribute forms:

Object (mai)

Object name	Arbitrary identifying value

Instance of Object (mai)

Object name	Arbitrary identifying value

Attribute for instance (mai)

Object name	Arbitrary identifying value	Attribute name

1mai indicates that the object has a multiple attribute identifier.

Attribute value (enumerated) for instance (mai)

Object name	Arbitrary identifying value	Attribute name	Attribute value

Attribute value (range) for instance (mai)

Object name	Arbitrary identifying value	Attribute name	Low value	High value

2.3 Examples of Bridge Tables for Data

Example 1. Object to Attribute Value for Instance, Attribute to Domain of Attribute

Client: an application involving dogs
Server: Architecture 4" (as described in Chapter 9 of *Object Lifecycles)*

Here, the objects in the client domain

Dog (Dog ID, Breed, Weight, etc., Dog Owner (R))[1]
Dog Owner (Owner ID, Address, etc.)

correspond to a Dog class and a Dog Owner class in the architectural domain. Similarly, each application attribute corresponds to an instance variable in that class. The architectural objects concerned are

Class (Class Name, Identifier Variable Name, . . .)
Instance Variable (Variable Name, Representation type, . . ., Class Name (R))

The required empty bridge tables are:

Table I

(client object becomes instance of Class)

Client		***Server***				
Object (sai)		Attribute value (enumerated) for instance (sai)				
Object name (a)	Identifying attribute name (b)	Object name (c)	Identifying attribute name (d)	Identifying attribute value (e)	Attribute name (f)	Attribute value (g)

Table II

(client attribute becomes instance of Instance

Client		***Server***		
Attribute		Domain of attribute (enumerated)		
Object name	Attribute name	Object name	Attribute name	Attribute value

1In this textual notation for an object together with its attributes (described in *Object-Oriented Systems Analysis: Modeling the World in Data*), the primary identifier is underlined.

Variable)

(h)	(i)	(j)	(k)	(l)

The instructions provided to the client by the server are as follows:

> For each object on the client IM,
>
> > In Table I put the object's name in columns a and e. Fill in the object's identifier in columns b and g. Fill in the words "class", "class name", and "identifying instance variable" in columns c, d, and f respectively.
> >
> > For each attribute of the object make an entry in Table II. The object's name is entered in column h and the attribute's name is entered in columns i and l. Fill in the words "instance variable" in column j and "variable name" in column k.
> >
> > From the instructions, we can see that all columns on the server side of the bridge table can be omitted: columns c, d, h, f, and k because each one contains a constant value. Similarly, columns e, g, and l can be omitted, since each one duplicates another in the same table.

The client now populates the tables:

Table I	***Client***		***Server***				
	Object (sai)		Attribute value (enumerated) for instance (sai)				
(client object becomes instance	Object name (a)	Identifying attribute name (b)	Object name (c)	Identifying attribute name (d)	Identifying attribute value (e)	Attribute name (f)	Attribute value (g)
of Class)	Dog	Dog ID	Class	Class name	Dog	Identifying instance variable	Dog ID
	Dog Owner	Owner ID	Class	Class name	Dog Owner	Identifying instance variable	Owner ID
	Food	Food ID	Class	Class name	Food	Identifying instance variable	Food ID

Table II	Client		Server		
	Attribute		Domain of attribute (enumerated) (sai)		
(client attribute becomes	Object name (h)	Attribute name (i)	Object name (j)	Attribute name (k)	Attribute value (l)
instance of	Dog	Dog ID	Instance variable	Variable name	Dog ID
Instance	Dog	Weight	Instance variable	Variable name	Weight
Variable)	Dog	Owner ID	Instance variable	Variable name	Owner ID
	Dog Owner	Owner ID	Instance variable	Variable name	Owner ID
	Dog Owner	Address	Instance variable	Variable name	Address
	. . .	. . .	Instance variable	Variable name	. . .

This bridge is statically populated. The information contained in these bridge tables is used to populate archetype code for the client prior to compilation.

Example 2. Attribute Value (range) to Instance of Object

Client domain: Any application
Server domain: Alarms

The particular alarm domain being considered here has the capability to inspect data values and to alert the operator when a data value is out of range. The client must supply the instance and attribute to be monitored and the high and low limits of the legal range. The alarm domain has an object

Monitored Analog Data Item (Data item ID, Low limit, High limit, <additional attributes>)

The required bridge table looks like this after population by the server:

Client	***Server***

Attribute value (range) for instance (sai)						Instance of Object (sai)		
Object name	Identifying attribute name	Identifying attribute value	Attribute name	Low limit	High limit	Object name	Identifying attribute name	Identifying attribute value
						Monitored analog data item	Data item ID	
						Monitored analog data item	Data item ID	
						Monitored analog data item	Data item ID	

The two shaded columns are eliminated because they are "constant value" columns.

After population of this table by the client we have:

Client						***Server***
Attribute value (range) for instance (sai)						Instance of object (sai)
Object name	Identifying attribute name	Identifying attribute value	Attribute name	Low limit	High limit	Identifying attribute value
Pump	Pump ID	105	Flow rate	200	210	
Pump	Pump ID	92A	Flow rate	150	155	
Power supply	Power supply ID	PS6	Voltage	140	141	
Magnet	Magnet ID	1	Current	416	418	

Magnet	Magnet ID	2	Current	273	275	
Power supply	Power supply ID	17	Voltage	130	131	

The last column contains an arbitrary identifier that is filled out by the server in its final population step.

In contrast to the previous example, this bridge is used (by the server) at run time. Furthermore, it is a dynamically populated bridge, in that instances may be added or deleted at the client's request.

Example 3. A full-sized data mapping example with multiple identifiers

Client: Biomedical application
Server: User Interface

This user interface domain has the capability of associating two different words specified by the client with a simple icon (square, circle, triangle, etc.). The color of the icon is given by one of these words, and the flash/no-flash property of the icon is determined by the other.

The client prepares for building the bridge by developing a sketch of the display he wants to define. This sketch is labeled with a coordinate system. On the sketch we can see three triangles located at (100,300), (500,300), and (600,300). The triangles are labeled INT1, INT2, and INT3, respectively. Notes accompanying the sketch indicate that each such triangle represents an integrator chassis and that the color of a triangle is to be green if the power supply associated with the integrator chassis is on, and red if it is off. Also, a triangle is to flash if the corresponding integrator chassis is saturated, and to be displayed without flash if the integrator chassis is not.

The server domain includes the following objects:

Window (<u>Window ID</u>, s low, s high, t low, t high)	Each window has a unique identifier and a screen position which defines its coordinate system
Display (<u>D ID</u>, x low, x high, y low, y high)	Each separate display has its own unique identifier and coordinate system
Display in Window (<u>D ID, Window ID</u>)	Any display may be placed in any window
Point Placed Icon (<u>PP Icon ID</u>, . . .)	Provides a repertoire of icons such as squares, rectangles, etc. These can be used on any display.
User PP Icon Type (<u>D ID (R), User icon name</u>, shapename(R))	Allows a user to establish a name (like pump, ion chamber, etc.) and a shape for an application icon type.
User PP Icon Instance (<u>D ID, User icon name, User PP icon instance</u>, x, y, label, color word, flash word)	Places a labeled application icon at user coordinates (x,y) on the display. The icon is colored and flashed as specified by the values of a color word and a flash word.

Color Policy (<u>D ID (R), User icon name</u>, color word value, color)	Establishes the policy to be used for coloring all instances of a user icon.
Flash Policy (<u>D ID (R), User icon name</u>, flash word value, flash)	Establishes the policy to be used for flashing all instances of a user icon.

Upon inspection of the preceding objects, it is apparent that the client must establish a unique display ID (D ID) and then supply data to populate Display, User PP Icon Type, User PP Icon Instance, Color Policy and Flash Policy. These tables must therefore form the basis of the bridge. Note that the client has already assembled the required information on the sketch and in the accompanying notes; all that is required now is to set up bridge tables that can be used to transcribe the client information into terms understood by the server. In a number of cases, these tables will not require a "client side": This happens because we are simply extending the mapping into the server sufficiently so that the server has all the required information.

For the Display object the server develops table A[1], which is easily populated by the client from the information on the sketch.

A

Server				
Attribute value (enumerated) for instance (sai)				
<u>Object name</u>	<u>Identifying attribute name</u>	<u>Identifying attribute value</u>	<u>Attribute name</u>	Attribute value
Display	D ID	103	x low	0
Display	D ID	103	x high	1000
Display	D ID	103	y low	0

1The first two columns are not shaded because we are assuming that the client will use these same bridge tables to define additional displays.

Display	D ID	103	y high	600

For User PP Icon Type, which has multiple attributes in the identifier, we produce two tables:

B

Server			
Identifier (mai)			
Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value
User PP Icon Type	1	D ID	103
User PP Icon Type	1	User icon type	integrator

C

Server			
Attribute value (enumerated) for instance (mai)			
Object name	Arbitrary identifying value	Attribute name	Attribute value
User PP Icon Type	1	shapename	triangle

For User PP Icon Instance, the server provides three more tables:

D

Server			
Identifier (mai)			
Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value
User PP Icon Instance	1	DID	103
User PP Icon Instance	1	User icon name	Integrator
User PP Icon Instance	1	User PP icon instance	1
User PP Icon Instance	2	DID	103
User PP Icon Instance	2	User icon name	Integrator
User PP Icon Instance	2	User PP icon instance	2
User PP Icon Instance	3	DID	103
User PP Icon Instance	3	User icon name	Integrator
User PP Icon Instance	3	User PP icon instance	3

E

Server			
Attribute value (enumerated) for instance (mai)			
Object name	Arbitrary identifying value	Attribute name	Attribute value
User PP icon instance	1	x	100
User PP icon instance	1	y	300

User PP icon instance	1	label	INT1
User PP icon instance	2	x	500
User PP icon instance	2	y	300
User PP icon instance	2	label	INT2
User PP icon instance	3	x	600
User PP icon instance	3	y	300
User PP icon instance	3	label	INT3

F

Client				*Server*		
Attribute for instance (sai)				Attribute for instance (mai)		
Object name	Identifying attribute name	Identifying attribute value	Attribute name	Object name	Arbitrary identifying value	Attribute name
Integrator power supply	Integrator power supply ID	monitor	on/off	User PP icon instance	1	color word
Integrator Chassis	Chassis ID	monitor	saturation	User PP icon instance	1	flash word
Integrator power supply	Integrator power supply ID	dose	on/off	User PP icon instance	2	color word
Integrator Chassis	Chassis ID	dose	saturation	User PP icon instance	2	flash word
Integrator power supply	Integrator power supply ID	backup	on/off	User PP icon instance	3	color word
Integrator Chassis	Chassis ID	backup	saturation	User PP icon instance	3	flash word

And finally, for Color Policy and Flash Policy:

G

Server			
Identifier (mai)			
Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value
Color policy	1	DID	103
Color policy	1	User icon name	integrator

Color policy	2	DID	103
Color policy	2	User icon name	integrator
Flash policy	1	DID	103
Flash policy	1	User icon name	integrator
Flash policy	2	DID	103
Flash policy	2	User icon name	integrator

H

Server			
Attribute value (enumerated) for instance (mai)			
Object name	Arbitrary identifying value	Attribute name	Attribute value
Color policy	1	color word value	1
Color policy	1	color	green
Color policy	2	color word value	0
Color policy	2	color	red
Flash policy	1	flash word value	0
Flash policy	1	flash	no flash
Flash policy	2	flash word value	1
Flash policy	2	flash	flash

Finally, note that we can make this set of bridge tables more compact by combining them wherever they have exactly the same column labels. Referring to the letters at the left of each of the foregoing bridge tables, we have:

Tables A and F: unchanged

Tables B, D, G:

Server			
Identifier (mai)			
Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value

User PP Icon Type	1	D ID	103
User PP Icon Type	1	User icon type	integrator
User PP Icon Instance	1	User PP icon instance	1
User PP Icon Instance	2	DID	103
User PP Icon Instance	2	User icon name	Integrator
User PP Icon Instance	2	User PP icon instance	2
User PP Icon Instance	3	DID	103
User PP Icon Instance	3	User icon name	Integrator
User PP Icon Instance	3	User PP icon instance	3
Color policy	1	DID	103
Color policy	1	User icon name	integrator
Color policy	2	DID	103
Color policy	2	User icon name	integrator
Flash policy	1	DID	103
Flash policy	1	User icon name	integrator
Flash policy	2	DID	103
Flash policy	2	User icon name	integrator

For Tables C, E, and H:

Server			
Attribute value (enumerated) for instance (mai)			
Object name	Arbitrary identifying value	Attribute name	Attribute value
User PP Icon Type	1	shapename	triangle
User PP icon instance	1	x	100
User PP icon instance	1	y	300
User PP icon instance	1	label	INT1
User PP icon instance	2	x	500
User PP icon instance	2	y	300
User PP icon instance	2	label	INT2
User PP icon instance	3	x	600
User PP icon instance	3	y	300
User PP icon instance	3	label	INT3
Color policy	1	color word value	1
Color policy	1	color	green
Color policy	2	color word value	0
Color policy	2	color	red

Flash policy	1	flash word value	0
Flash policy	1	flash	no flash
Flash policy	2	flash word value	1
Flash policy	2	flash	flash

3. Algorithm

3.1 Bridge Tables for Algorithm

To define algorithmic elements in the bridge, we make a correspondence between algorithmic elements in the client and algorithmic elements in the server. The standard half-tables used for this purpose are as follows:

Action

Object name	State number

Process

Process ID	Process name

Formal Parameters of Process

Process ID	Parameter number[1]	Parameter name	Parameter type

Actual Parameters of Process

Process ID	Parameter number	Parameter value	Parameter type

1Because these are formal parameters, only Process ID and Parameter Name are required to make up an identifier.

3.2 Examples of Bridge Tables for Algorithms

Example 1: Client Accessor to Architectural Process

Client: Juice plant application
Architecture: a table-based architecture

This architecture is based on the following objects:

Table (<u>Table name</u>, Identifier Column name, Number of rows)
Column (<u>Table name, Column name</u>, Implementation data type, Start address)

The Table table is derived from objects in the various domains (including the application); the Column table is derived from attributes. The tables have already been populated by data bridge tables, as follows:

Client		***Server***				
Object (sai)		Attribute value (enumerated) for instance (sai)				
<u>Object name</u>	Identifier attribute name	<u>Object name</u>	<u>Identifying attribute name</u>	<u>Identifying attribute value</u>	<u>Attribute name</u>	Attribute value
Batch	Batch_ID	Table	Table_name	Batch	Column_name	Batch_ID
Cooking_Tank	Tank_ID	Table	Table_name	Cooking_Tank	Column_name	Tank_ID
. . .	. . .	Table	Table_name		Column_name	

Server			
Identifier (mai)			
<u>Object name</u>	<u>Arbitrary identifying value</u>	<u>Identifying attribute name</u>	Identifying attribute value

Column	1	Table name	Cooking_Tank
Column	1	Column name	Capacity
Column	2	Table name	Heater
Column	2	Column name	On/off
Column	3	Table name	Cooking_Tank
Column	3	Column name	Actual_temperature

Server			
Attribute value (enumerated) for instance (mai)			
Object name	Arbitrary identifying value	Attribute name	Attribute value
Column	1	implementation data type	real
Column	2	implementation data type	boolean
Column	3	implementation data type	real

Note that this architecture requires that each table have a single column that can be used as an identifier.

The architecture also provides a function called select_1_attr. Client domains can invoke this function providing the name of the table, the value of the identifier of the instance in which the client is interested, and the name of the attribute whose value the client wishes to obtain. Select_1_attr returns the value of the specified attribute. That is,

select_1_attr (in: table, ident_val; out: attr_val)

The purpose of this bridge is to generate code from the ADFDs of the client. Accordingly, the architecture constructs the following bridge table for the client:

Client				***Server***			
Actual parameters of process				Formal parameters of process			
Process ID	Parameter number	Parameter value	Parameter type	Process ID	Parameter number	Parameter name	Parameter type

The instructions provided to the client are:

For each invocation of select_1_attr, make two entries in the Table.

Entry 1:

- Enter the process ID in column 1.
- Enter the number 1 in column 2
- Enter the object name (with blanks replaced by underscores) in column 3.

Entry 2:

- Enter the process ID in column 1
- Enter the number 3 in column 2
- Enter the output attribute name (with blanks replaced by underscores) in column 3

What the server has in mind is shown next:

Client	**Server**
Actual parameters of process	Formal parameters of process

Process ID	Parameter number	Parameter value	Parameter type	Process ID	Parameter number	Parameter name	Parameter type
B.1	1	Batch		select_1_attr	1	table name	character
B.1	2	Batch_ID		select_1_attr	2	ident_val	character
B.1	3	Tank_ID		select_1_attr	3	attr_value	character

However, the server recognizes that all the shaded information can be determined from information already available (the value of ident_var can be determined from the value of table; this was given in the Table table). In addition, the run-time typing of ident_val and the attr_val can be determined from the previously populated Column table. This is a little secret the server hasn't shared with us: This server actually has several versions of select_1_attr, and it intends to generate the appropriate invocation based on the run-time typing of the actual parameters.

The client (the juice plant application) now populates a reduced form of the preceding table. Referring to the figures on pages 116, 119, and 120 of *Object Lifecycles*[1], we obtain the following result:

Client		
Actual parameters of process		
Process	Parameter number	Parameter value
B.1	1	Batch
B.1	3	Tank_ID
CT.1	1	Cooking_Tank
CT.1	3	Actual_temperature

1... and ignoring the process TR.5 on page 119, which is clearly *not* the same as process TR.5 on page 120 . . .

CT.2	1	Cooking_tank
CT.2	1	Cooking_tank
TR.5	1	Temperature_ramp
TR.5	3	Batch_ID

This example illustrates the mapping of a read accessor into the architecture. Other read accessors can be treated similarly, as can write and delete accessors.

Example 2: Context-Free Processes.

Client: Biomedical Application
Server: Architecture

The architecture used for the Biomedical system provided a number[1] of context-free processes, such as

Calculate area of annulus (in: inner radius, outer radius; out: area)
Calculate prism volume (in: area, height; out: volume)
Test limits (in: data value, high limit, low limit; out: ok/not ok)

The architecture contains the following objects:

Function (<u>function name</u>)
Formal parameter (<u>function name, formal parameter name</u>, parameter type)

1About 40 or 50, as I recall. All word unpacking and conversion in the PIO domain was accomplished by this method, as well as all computation for the application domain.

populated as follows:

Function name
calculate area of annulus
calculate prism volume
test limits
. . .

Function name	Formal parameter name	Parameter type
calculate area of annulus	inner radius	real
calculate area of annulus	outer radius	real
calculate area of annulus	area	real
calculate prism volume	area	real
calculate prism volume	height	real
calculate prism volume	volume	real
test limits	data value	real
test limits	low limit	real
test limits	high limit	real
test limits	ok status	boolean

Test and transformation processes (the only context-free processes on an ADFD) are mapped into the architectural processes by means of the following bridge table provided by the architecture, and populated by the client:

Client		***Server***		
Process		Instance of Object (sai)		
Process ID	Process name	Object name	Identifying attribute name	Identifying attribute value
R.1	Compute central area	Function	Function name	Calculate annulus area
R.2	Compute ring area	Function	Function name	Calculate annulus area
R.10	Compute ring volume	Function	Function name	Calculate prism volume
PS.3	Check power supply voltage	Function	Function name	Test limits
R.6	Check central ring dose	Function	Function name	Test limits

Server			
Identifier (mai)			
Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value
Formal parameter	1	Function name	calculate area of annulus
Formal parameter	1	Formal parameter name	inner radius
Formal parameter	2	Function name	calculate area of annulus
Formal parameter	2	Formal parameter name	outer radius
Formal parameter	3	Function name	calculate area of annulus
Formal parameter	3	Formal parameter name	area
Formal parameter	4	Function name	calculate prism volume
Formal parameter	4	Formal parameter name	area
Formal parameter	5	Function name	calculate prism volume
Formal parameter	5	Formal parameter name	height
Formal parameter	6	Function name	calculate prism volume
Formal parameter	6	Formal parameter name	volume
Formal parameter	7	Function name	test limits
Formal parameter	7	Formal parameter name	data value
Formal parameter	8	Function name	test limits

Formal parameter	8	Formal parameter name	low limit
Formal parameter	9	Function name	test limits
Formal parameter	9	Formal parameter name	high limit
Formal parameter	10	Function name	test limits
Formal parameter	10	Formal parameter name	ok status

Client					***Server***		
Actual parameters of process					Instance of object (mai)		
Process ID	Process name	Param-eter number	Parameter value	Param-eter type	Object name	Arbi-trary identify-ing value	Attri-bute value
R.1	compute central area	1	0.	real	Formal param-eter	1	real
R.1	compute central area	2	ring (1) .outer radius	real	Formal param-eter	2	real
R.1	compute central area	3	ring (1).area	real	Formal param-eter	3	real
R.2	compute ring area	1	ring(k). inner radius	real	"	1	real
R.2	compute ring area	2	ring(k). outer radius	real	"	2	real
R.2	compute ring area	3	ring(k). area	real	"	3	real
R.10	compute ring volume	1	ring(k). area	real		4	real

R.10	compute ring volume	2	in-service ion chamber(k).foil sep	real		5	real
R.10	compute ring volume	3	ring(k). volume	real		6	real
PS.3	check power supply voltage	1	integrator power supply(j). voltage	real		7	real
PS.3	check power supply voltage	2	integrator power supply(j). low limit	real		8	real
PS.3	check power supply voltage	3	integrator power supply(j). high limit	real		9	real
PS.3	check power supply voltage	4	ok status	boolean		10	bool-ean
R.6	check central ring dose	1	ring (1).dose	real		7	real
R.6	check central ring dose	2	0.	real		8	real
R.6	check central ring dose	3	patient.prescribed dose	real		9	real
R.6	check central ring dose	4	not yet achieved	boolean		10	bool-ean

3.4 Algorithms Arising Because of the Bridge

When building a bridge, one may become aware of a previously unidentified piece of processing. The classic example is that of a train tracking application - user interface bridge. The trackage, along which any train must travel, is depicted schematically on a display as a series of connected zig-zag lines. Associated with each end of a line segment is (a) a user display coordinate and (b) a milepost number -- essentially the distance from the beginning of the trackage. Hence we have this idea:

	User coordinates	
Milepost	x	y

0.00	100	100
2.36	300	100
4.82	421	372
7.00	665	436
. . .	. . .	. . .

Now, as a train travels along the tracks, the application (client) knows the position of the train in terms of mileposts. The application would like to show the train as a moving icon -- moving to the position on the screen that corresponds to its milepost position. The question is this: to what domain[1] do we assign the process that computes the proper screen coordinates given the train's milepost position?

I assert that the answer is based in reusability: if there is any possible situation in which similar functionality could be required for a different client, the processing should be associated with the more general (server) domain.

In this particular case, it is easy to come up with applications that require the same or similar processing:

- Display of an aircraft traveling between waypoints (air traffic control)
- A highway emergency dispatch application where accidents are anecdotally reported in terms of posted mileposts and observable roadway features (" . . . traveling west having passed milepost 24.2 and before the intersection with Geschwindt road")
- A naval warfare application that includes a display showing the position (latitude and longitude) and type of each ship in the relevant area.

Therefore, in this particular case, one would assign a function "convert world coordinates to display coordinates" to the server domain.

Such a discovery -- the need to place, move, or delete icons on a display -- may well have additional implications. For the projected uses given here, it is clear that the server will

1In accordance with assumption 4 that states that "a bridge is similar to a clear pane of glass: It contains . . . nothing", the processing cannot be associated with the bridge. Therefore it must be assigned to one of the participating domains.

need consequent internal functionality: the ability to repaint the background when an icon is moved or removed. Such consequent functionality should also be considered when making the decision as to which domain to assign the algorithm that arose during bridge building.

4. Control

4.1 Control Mappings

The purpose in specifying the control aspects of a bridge is to make a correspondence between invocations in one domain and executions in another domain. Those invocations that appear explicitly on the client (or server) models take the form of event generation[1]. The corresponding executions take the form of either reception of a direct transfer of control to a process or of event reception by an action.

Note that the issue in control mappings is not synchronization (as one might first expect), but of *order of execution*. That is, an invocation that appears to be synchronous can, in fact, be mapped to an asynchronous invocation, and an asynchronous invocation can be mapped to a synchronous one -- so long as the assumptions made by the analysis of the participating domains are maintained. Primary among such assumptions are these:

- Any ordering of execution specified by the analysis must be preserved.
- Any action within a domain must proceed to completion before any interfering action is initiated.

Even the simplest case is subtly treacherous: Suppose an object generates several events in a given action. For simplicity, assume that these events are all targeted at objects in the same domain as the generating object. Because, in general, the processes on the ADFD are not entirely sequenced, we must allow for the possibility that some write accessors remain to be executed after the first event is generated, and that such a write accessor will record data that will be read by an action triggered by the event that has been generated. There is clearly the possibility of a race condition here, and, of course, we must understand and control -- or eliminate -- the race.

One solution is to move the responsibility to the architectural domain (where it belongs anyway), arranging it so that every event is queued when generated and that control is immediately returned to the generating action so that the action can complete. Only then can the architecture deliver another event to one of its clients.

Another solution requires careful examination of the analysis thread of control. If you can arrange to have all accessors, transformations, and tests execute completely before any event generators, you can implement an event generator as a synchronous transfer of control to the action that needs to receive the event (as was done in the architecture of Chapter 9 of Object Lifecycles). This gives you complete control over the potential race

1Implicit invocations were dealt with in Section 3.2 Bridge Example for Algorithms condition.

What does this have to do with choosing control mappings? Exactly nothing! We were motivated to go into this matter fairly extensively here to lay to rest the common misconception that in building a control mapping one is making a statement about how control is to be transferred -- synchronously or asynchronously. This decision is the responsibility of the architectural domain. The control mappings, taken together with the ADFDs, serve to inform the architectural domain of the ordering of execution requirements across the entire system.

4.2 Bridge Tables for Control

The following forms are provided to specify the half-tables of the bridge:

Event (sai)

Event ID	Identifying attribute name

Event (mai)

Event ID	Arbitrary identifying value

Formal parameters of an event

(recipient's

view)

Event ID	Parameter number	Parameter name	Parameter type

Actual parameters of an event

(generator's

view)

Event ID	Parameter number	Parameter value	Parameter type

As usual, the use of these half tables will be illustrated by a series of examples.

4.3 Control Mapping Examples

Example 1: Event - event mapping.

Client: Application or PIO
Server: Alarm Service

The client generates various events that are directed ultimately at the operator. These events have the nature of an alarm.

The particular alarm service domain considered here includes the following objects:

Alarm Spec (<u>Alarm Spec ID</u>, alarm text, priority, number of parameters)	Establishes the repertoire of alarm types in terms of text to be presented to the operator and priority of this alarm type
Alarm Text Formal Parameters (<u>Alarm Spec ID, Parameter number</u>, Parameter type)	
Alarm Incident (<u>Incident ID, Alarm Spec ID(R)</u>, time of occurrence)	An occurrence of an alarm of the specified type
Alarm Incident Actual Parameters (Incident ID (R), <u>Alarm Spec ID (R), Parameter number</u>, Parameter value)	

The alarm service domain expects to receive events of the following form from its clients:

AI1: Alarm Incident Occurred (Alarm Spec ID, param2, param3, . . .)

The alarm service domain is then expected to display the alarm text to the operator.

Embedded within the alarm text are various fields referring to the time of occurrence and to the event parameters; these embedded fields tell how the information carried by the event is to be displayed. For example, if the alarm text for Alarm Spec 22 is

%T Train %1I %3I minutes late at %2C station

and we receive the event

Alarm Incident occurred (22, 301, Berkeley, 4)

the operator is to see

09:10:03 Train 301 4 minutes late at Berkeley station

The server provides several bridge tables for the client.

To define the client's repertoire of alarms (an off-line function):

A

Client	***Server***				
Event (sai)	Attribute value (enumerated) for instance (sai)				
<u>Event</u>	<u>Object name</u>	<u>Identifying attribute name</u>	<u>Identifying attribute value</u>	<u>Attribute name</u>	Attribute value
	Alarm Spec	Alarm Spec ID	=f(Event ID)	Alarm text	
	Alarm Spec	Alarm Spec ID	=f(Event ID)	Priority	
	"	"	"	Number of parameters	
	"	"	"	Alarm text	
				Priority	
	"	"	"	Number of parameters	

Once the client has populated this table, the server will provide a value for Alarm Spec ID based upon the client's event number.

To define the Alarm Spec Formal Parameters:

B	**_Server_**			
	Identifier (mai)			
	Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value
	Alarm text Formal Parameter	1	Alarm Spec ID	= f(Event ID)
	Alarm text Formal Parameter	1	Parameter number	
	Alarm text Formal Parameter	2	Alarm Spec ID	= f(Event ID)
	Alarm text Formal Parameter	2	Parameter number	
	Alarm text Formal Parameter	3	Alarm Spec ID	= f(Event ID)
	Alarm text Formal Parameter	3	Parameter number	
	Alarm text Formal Parameter	4	Alarm Spec ID	= f(Event ID)
	Alarm text Formal Parameter	4	Parameter number	
	"	. . .	. . .	. . .

C	*Client*			*Server*			
	Actual parameters of an event			Attribute value enumerated for instance (mai)			
	Event ID	Actual parameter number	Parameter type	Object name	Arbitrary identifying value	Attribute name	Attribute value
				Alarm spec formal parameter	1	parameter type	=parameter type
				Alarm spec formal parameter	2	parameter type	=parameter type
				Alarm spec formal parameter	3	parameter type	=parameter type
				"	. . .	"	"

The server's instructions to the client are:

For each event to be directed to the alarm service domain

Make three entries in Table A to specify the required alarm text, priority, and the number of parameters for the alarm to be associated with event occurrences of this type

For each item of event data:

- make 1 entry in Table B to assign a parameter number. Number the parameters starting with 2.
- make 1 entry in Table C to supply the parameter type of the parameter specified in Table B.

The server then completes the population of these tables by assigning a unique value for each Alarm Spec ID.

The Alarm Incident and Alarm Incident Actual Parameters objects are populated at run time whenever the alarm service domain receives an AI1 event from the client. To cause the client to generate AI1 events properly, the server provides archetype code that is populated from the following bridge table.

D	*Client*	*Server*

Actual parameters of event				Formal parameters of event			
Event ID (a)	parameter number (b)	parameter value (c)	parameter type (d)	Event ID	parameter number	parameter name	parameter type
				AI1	=(b)	param<b>	=(d)
				AI1	=(b)	param<b>	=(d)
				AI1	=(b)	param<b>	=(d)
				AI1	=(b)	param<b>	=(d)

This information, together with the correspondence between event numbers and alarm spec IDs (Table A) is sufficient to populate the archetype code for the client.

Example 2: Event to Invocation

Client domain: Application
Server domain: PIO

The client's ADFDs show the generation of various events that are intended to turn pumps on and off and to open and close valves. The client assumes that the PIO domain will receive these events and perform the necessary manipulations of the external equipment.

The PIO domain includes the objects

Digital out point (DOP ID, OR ID(R), . . .)

Output Rule (OR ID, User value, Hardware value)

The data aspect of the bridge has already been established through the following tables:

A

Client			***Server***				
Instance of Object (sai)			Attribute value (enumerated) for object (sai)				
Object	Identifying	Identifying	Object	Identifying	Identifying	Attribute	Attribute

name	attribute name	attribute value	name	attribute name	attribute value	name	value
Pump	Pump ID	1	Digital output point	DOP ID	26	OR ID	12
Pump	Pump ID	2	Digital output point	DOP ID	38	OR ID	12
. . .	. . .	. . .	"	"			. . .
Pump	Pump ID	25	Digital output point	"	100	"	12
. . .	. . .	. . .	"	"			. . .
Valve	Valve ID	103A	"		110		19
Valve	Valve ID	103B	"		111		20
Valve	Valve ID	96	"		120		19

B

Server			
Identifier for object (mai)			
Object name	Arbitrary identifying value	Identifying attribute name	Identifying attribute value
Output rule	1	OR ID	12
Output rule	1	User value	on
Output rule	2	OR ID	12
Output rule	2	User value	off
Output rule	3	OR ID	19
Output rule	3	User value	open
Output rule	4	OR ID	19
Output rule	4	User value	closed
Output rule	5	OR ID	20
Output rule	5	User value	open
Output rule	6	OR ID	20
Output rule	6	User value	closed

C

Client	**Server**
Domain of attribute (enumerated)	Attribute value (enumerated) for instance (mai)

Object name	Attribute name	Attribute value	Object name	Arbitrary Identifying value	Attribute name	Attribute value
Pump	Desired State	on	Output rule	1	Hardware value	1
Pump	Desired State	off	Output rule	2	Hardware value	0
Valve	Desired State	open	Output rule	3	Hardware value	0
Valve	Desired State	closed	Output rule	4	Hardware value	1
Valve	Desired State	open	Output rule	5	Hardware value	1
Valve	Desired State	closed	Output rule	6	Hardware value	0

The PIO domain also provides a function

digout (in: DOP ID, user value)

The events shown on the client's ADFDs are:

E21: turn on pump (pump ID)
E22: turn off pump (pump ID)
E23: open valve (valve ID)
E24: close valve (valve ID)

Note that the client has chosen to define the events so that the object (pump or valve) and desired state are encoded in the event label.

To map these events to digout, we develop the following bridge table:

Client				***Server***			
Actual parameters of event				Formal parameters of process			
Event number	Parameter number	Parameter value	Parameter type	Function	Parameter number	Parameter name	Parameter type
E21	1	pump ID	integer	digout	1	DOP ID	integer
E21	2	on	integer	digout	2	user value	integer
E22	1	pump ID	integer	digout	1	DOP ID	integer
E22	2	off	integer	digout	2	user value	integer
E23	1	valve ID	integer	digout	1	DOP ID	integer
E23	2	open	integer	digout	2	user value	integer
E24	1	valve ID	integer	digout	1	DOP ID	integer
E24	2	closed	integer	digout	2	user value	integer

The value of DOP ID can be filled in from Table A.

It is the intention of the server to use this table to populate archetype code to generate code for the client. These invocations so generated will be used in place of the event generators that generate E21, E22, E23, and E24. Taking into account the information contained in Table A, the server now has all of the information required to populate the archetype. In addition, the server has enough information to execute digout when invoked: Given DOP ID, digout can look up OR ID from the Digital Output Point table. Then, by looking in the Output Rule table, digout can find the hardware value to send to the external equipment.

5. Bridges that Mix Data, Algorithm, and Control

Typically, a bridge between domains maps data elements to data elements, algorithmic elements to algorithmic elements, and control elements to control elements. However, in Section 4.3, example 1, we found that, in order to support an event-event mapping, we had to supply also an event-data mapping. This is, in fact, quite normal; do not hesitate to define such "cross-over" mappings where appropriate.

Appendix:
Standard Forms for Bridge Tables

Data Forms: sai and mai

Attribute

<u>Object name</u>	<u>Attribute name</u>

Domain of attribute (enumerated)

<u>Object name</u>	<u>Attribute name</u>	<u>Attribute value</u>

Domain of attribute (range)

<u>Object name</u>	<u>Attribute name</u>	Low value	High value

sai Data Forms

Identifier (sai)

<u>Object name</u>	<u>Identifying attribute name</u>

Object (sai)

<u>Object name</u>	Identifying attribute name

Instance of Object (sai)

<u>Object name</u>	<u>Identifying Attribute name</u>	<u>Identifying Attribute value</u>

Attribute for instance (sai)*

<u>Object name</u>	<u>Identifying attribute name</u>	<u>Identifying attribute value</u>	<u>Attribute name</u>

Attribute value (enumerated) for instance (sai)

<u>Object name</u>	<u>Identifying attribute name</u>	<u>Identifying attribute value</u>	<u>Attribute name</u>	Attribute value

* This form is derived by joining Instance of Object (sai or mai) with Attribute.

Attribute value (range) for instance (sai)

Object name	Identifying attribute name	Identifying attribute value	Attribute name	Low value	High value

mai data forms

Identifier (mai)

Object name	Arbitrary identifying value	Identifying Attribute name	Identifying Attribute value

Object (mai)

Object name	Arbitrary identifying value

Instance of Object (mai)

Object name	Arbitrary identifying value

Attribute for instance (mai)*

<u>Object name</u>	<u>Arbitrary identifying value</u>	<u>Attribute name</u>

Attribute value (enumerated) for instance (mai)

<u>Object name</u>	<u>Arbitrary identifying value</u>	Attribute name	Attribute value

Attribute value (range) for instance (mai)

<u>Object name</u>	<u>Arbitrary identifying value</u>	<u>Attribute name</u>	Low value	High value

Algorithmic Forms

Action

<u>Object name</u>	<u>State number</u>

Process

<u>Process ID</u>	Process name

Formal Parameters of Process

<u>Process ID</u>	Parameter number	<u>Parameter name</u>	Parameter type

Actual Parameters of Process

<u>Process ID</u>	<u>Parameter number</u>	Parameter value	Parameter type

Control Forms

Event (sai)

<u>Event ID</u>	Identifying attribute name

Event (mai)

Event ID	Arbitrary identifying value

Formal parameters of event
(event receiver's view)

Event ID	Parameter number	Parameter name	Parameter type

Actual parameters of an event
(event generator's view)

Event ID	Parameter number	Parameter value	Parameter type