Bridges and Wormholes

Sally Shlaer
Stephen J. Mellor

5 August 1996

A bridge is a connection between two domains that may be used for transfer of control between the domains at run time. This paper describes how such transfers of control are specified in Shlaer-Mellor models and explains how the domains are connected when the system is constructed.

1. Assumptions and Rationale

Domain Replacement. Two very important goals of the Shlaer-Mellor method are (1) the ability to re-use an existing domain without modification and (2) the ability to replace one domain with another that accomplishes the same mission. The schemes we use for building models and bridges must take these goals into account.

One clear implication is that no information about one domain may ever appear in the models of another domain. That is, we must be able to establish the connections between domains not as part of the analysis process, but rather as part of the system construction process.

A more subtle implication has to do with how we think about the interface of a domain. Consider, for example, a situation wherein an application domain wishes to obtain the current value of a sensor-based attribute (Figure 1.1) from the Process Input/Output (PIO) domain. Depending on the properties of the particular PIO domain employed in the system, this functionality may be provided within the lifecycle of some PIO object (and so PIO is prepared to accept an external event -- an event from outside the domain -- to access the capability), or the functionality may be provided in the form of a synchronous service. Our scheme for building bridges must therefore allow the application model to connect correctly to PIO, regardless of how the required functionality is expressed in the PIO model we are currently using.

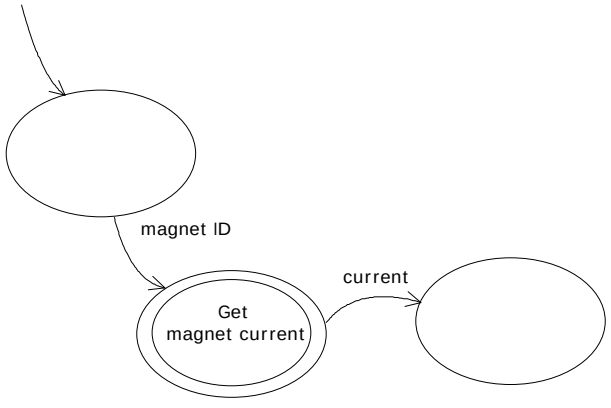


Figure 1.1: An application domain acquires a sensor-based data item from the PIO domain.

Representative Scenarios. Let us examine some scenarios in order to understand the essential requirements on transferring control across domain boundaries. Suppose that we have two domains named Home and Away. Suppose further that, on some ADFD in the Home domain, an invocation of a

wormhole process appears. The purpose of the invocation is to request a service from Away. The effect of the invocation is to cause a transfer of control into the Away domain, as shown in Figure 1.2.

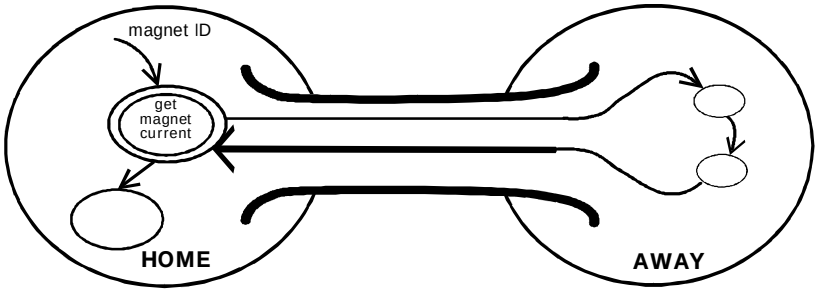


Figure 1.2: After requesting a service from Away, Home regains control and continues the action (informal sketch).

What happens next depends on the semantics of both domains. If Home requires no further communication from Away with respect to the requested service, control is simply returned to Home in the ADFD or SDFD in which the *request wormhole* was invoked. This return of control appears to Home to be synchronous with respect to the invocation of the wormhole. Additional activity may continue if necessary in Away to complete the service.

If, however, Home has requested that Away notify him (Home) at some point in the future, a more interesting scenario develops, as shown in Figure 1.3. Here a thread of control continues in Away. In due course, this thread of control reaches the point that it is appropriate to notify Home. This is done by Away by invoking a wormhole (in Away) that effects an "asynchronous return." From Home's perspective, this return of control occurs asynchronously with respect to the original wormhole process invocation: The asynchronous return takes the form of an external event directed to Home.

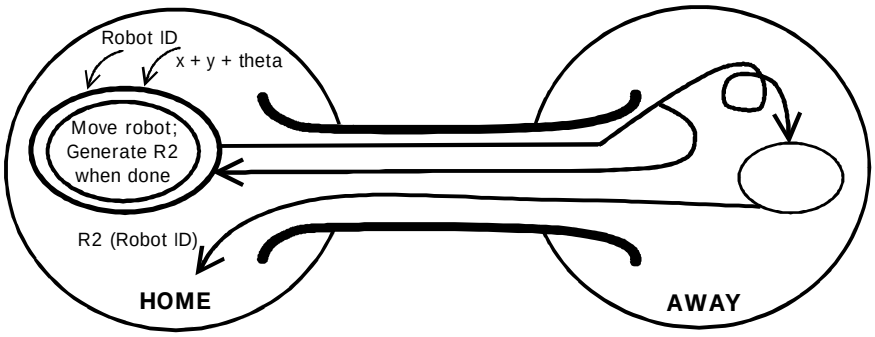


Figure 1.3: Home requests and receives control after the service is complete (informal sketch)

If desired, the asynchronous return may be made repeatedly. In Figure 1.4, Away generates the return every time the boiler pressure exceeds the stated limit.

Note that we are able to describe these scenarios without specifying the direction of the client-server relationship between Home and Away. That is, either domain may initiate a scenario by transferring control to the other, regardless of the direction of the bridge on the domain chart.

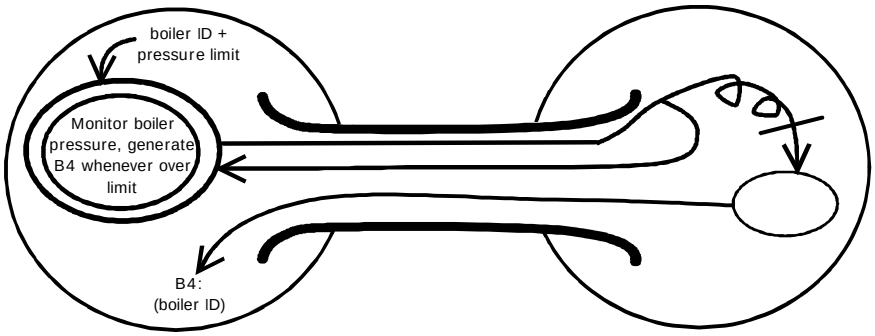


Figure 1.4: Home receives repeated notifications that the service has completed (informal sketch)

Bridge as a clear pane of glass. 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 another domain.

While there is no code in the bridge that can be associated with any particular domain, there can, in fact, be generic code. The purpose of this code is only to make the required correspondences. Because such correspondences can always be expressed in data, we assign responsibility for the bridge code to the architecture domain—the ultimate manager for all data in the system.

Semantic Shift. Data elements that appear on an ADFD or SDFD in a particular domain are defined in that domain only. Hence when a data element is passed into a wormhole, it re-appears in the receiving domain with a different name—one defined in the receiving domain. In addition, the type of data may change as it crosses a domain boundary: What appears as a temperature in one domain may appear as a real in another. The shift in type must be compatible, so we can map a data element of type temperature to one of type real but not to one of type Boolean. We call this change in name and type a 'semantic shift.'

2. Transfer Vectors and Asynchronous Returns

In the situations shown in Figures 1.3 and 1.4, the nature of the initial wormhole invocation included providing the Away domain with information to allow Away to communicate with Home at a later point. At any place in the models where control is to be returned across a domain boundary asynchronously and at some future time, we have the concept of a *transfer vector*: the information required to effect the transfer. In order to retain the greatest flexibility for the system architecture, the structure of a transfer vector is hidden in the formalism and architecture. The analyst needs to be aware of only the information content of the vector.

Sending a Transfer Vector. When the Home domain requires that Away return control apparently asynchronously to Home at some point in the future, Home must supply a transfer vector to Away. The transfer vector is based on an event defined in the Home domain. The analyst can think of the transfer vector as a partial event (an event label and an instance identifier only) that will be filled out with supplemental data (if such is defined for the base event) and returned to Home as a complete event at some future time.

For example, in Figure 1.3, Home provides Away with a transfer vector that contains the event label R2: Robot Move Complete and the instance identifier Robot ID. The event label is inherent in the meaning of the wormhole; the Robot ID is provided as input to the wormhole.

Receiving a transfer vector. When Away receives a transfer vector from Home, Away regards the transfer vector as a data element of type 'transfer vector.' Away must save the transfer vector for later use. This is done by attributing the transfer vector to an object that acts as a surrogate for the thread of control in the sending domain.

In Figure 1.3, the Away domain—here PIO—must save the transfer vector so that when the robot move completes, PIO may send an event to the Home domain. Similarly, in Figure 1.4, the value-monitoring domain (Away) must save the transfer vector in a Monitor Request or similar object.

Returning via transfer vector. When it is time for Away to provide the asynchronous notification to Home, Away invokes an asynchronous return wormhole, supplying as input data:

- the previously saved transfer vector
- any additional data elements to be returned to the calling domain (Home). These will be combined with the transfer vector as supplemental data items to form the event expected by Home.

The asynchronous return wormhole acts as a way to "return via transfer vector" back to the Home domain. An example of such a wormhole is shown in Figure 2.1. Notification of completion is then seen, in Home, as an external event to a specified instance in the normal manner—the 'asynchronous return' of the previous section. It is the responsibility of the architecture domain to package the input data elements as supplemental data items to the event specified by the transfer vector (in whatever form the architecture uses to reflect the concept of event), and to propagate the resultant event to the Home domain.

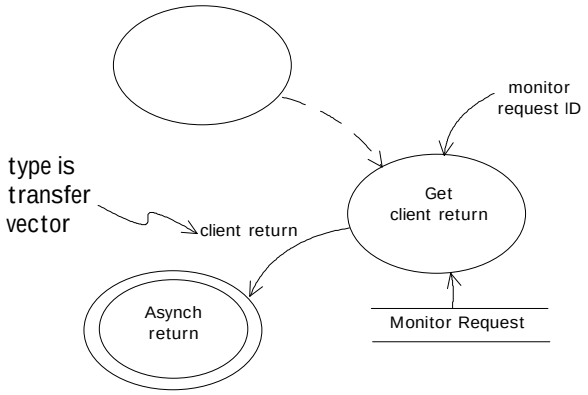


Figure 2.1: The value monitoring domain returns control via a transfer vector using an asynchronous return wormhole.

The asynchronous return wormhole usually appears in an ADFD in Away, as implied by the idea of returning control at some future time. However, there is nothing in the formalism to preclude providing an asynchronous return from an SDFD. If, for example, the functionality required by the 'Move Robot' wormhole were provided by a synchronous service, and if the robot moved essentially instantaneously, then it would be entirely appropriate for the synchronous service to provide the asynchronous return indicating Robot move complete.

3. Return Coordinates and Synchronous Returns

Now let us take a deeper look at synchronous returns. First, recall that synchronous and asynchronous returns have entirely different semantics:

- An asynchronous return returns control via an external event as specified by the request wormhole. This event will, in general, cause execution of some action other than the action containing the request wormhole.

- A synchronous return returns control to the ADFD or SDFD from which a request wormhole was invoked.

After a request wormhole has been invoked, control must be returned to the ADFD or SDFD containing the wormhole. This is necessary to allow the ADFD/SDFD to complete. The point to which control must be returned is known as a *return coordinate*. The structure of a return coordinate, like a transfer vector, is hidden in the formalism and in the architecture.

An analyst working in the Home domain—where the request wormhole lies—has no visibility of the return coordinate other than that implied by the connectivity of the ADFD or SDFD. The analyst for Away, on the other hand, has direct visibility of the return coordinate in only certain cases (as described below); in these cases he or she can think of the return coordinate as a simple return address.

Away's need to know about return coordinates is, surprisingly enough, determined by the data outputs shown for the request wormhole in Home.

Case 1. If a request wormhole shows no data outputs, the OOA rules of execution allow control to continue in Home's ADFD or SDFD as soon as the wormhole has been invoked. In this case the thread of control splits upon invocation of the wormhole, with one leg continuing in Home and the other in Away. Because this is an execution property of the formalism, the architecture must take the responsibility for creating and managing this fork in the thread. As a result, when the request wormhole has no data outputs, Away needs to do nothing to return control to Home: the architecture has already taken care of this by the time Away receives control.

Case 2. If a request wormhole does have data outputs, control cannot be returned to the invoking ADFD or SDFD until the required data has been assembled. Only the Away domain can determine when this has been done. Hence Away must make the synchronous return explicit; this is done by means of a synchronous return wormhole as shown in Figure 3.1.

Because, in general, (1) there can be multiple "request threads" active in the Away domain, (2) an action or synchronous service in Away can invoke request wormholes to still other domains, and (3) the formalism has no direct way of tracking which thread must lead to which synchronous return, we must place the responsibility for maintaining such connections in Away. This is done in exact parallel with the treatment of transfer vectors:

- When Away receives control, either via an external event or a synchronous service call, an input parameter of type return coordinate is provided. It is the responsibility of the architecture to provide a value for this data item in whatever form the architecture chooses to implement the concept.

- Away must save the return coordinate for later use by attributing the return coordinate to an object acting as a surrogate for Home's thread of control.

- When data values have been obtained for all of Home's synchronous outputs, the action or synchronous service then in execution must invoke a synchronous return wormhole. The inputs to this wormhole are the return coordinate and the data items to be returned to Home.

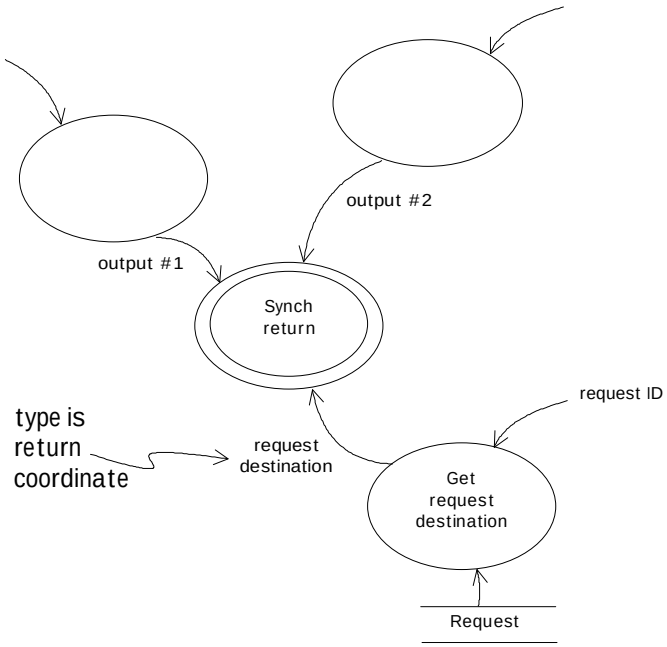


Figure 3.1: The PIO domain returns control via a return coordinate.

4. Interface of a Domain

Defining the Interface. When a domain receives control from across a domain boundary, the receiver sees the control as either the arrival of an event from outside the domain or as the invocation of a synchronous service. The complete (dynamic) interface of a domain may therefore be described by specifying the external events and the synchronous service interfaces of the domain, as follows:

- For each external event, specify the event label and meaning as defined in the recipient domain.
- For each synchronous service, specify the identifier of the synchronous service and its meaning.
- For each synchronous service or external event, describe the functionality provided. This description is similar in nature to a traditional procedure or function description.
- For each event data item and for each input or output parameter of a synchronous service, specify the meaning and data type as defined in the recipient domain.

Domain Interface Document. The domain interface is documented in a form useful to the architects and system construction specialists who must develop the connections between domains. A form we have found convenient for the Domain Interface Document is given below.

Synchronous Service

S1 Read AIP (in: AIP ID, return coordinate; out: scaled value)
Reads the value of the analog input point specified by AIP ID, returns the scaled (engineering unit) value.

External Events

AIOP1: Set Analog Input Point (AIP ID; setpoint)

Sets the analog input point specified by AIP ID to the specified setpoint (engineering unit) value. This operation is legal only for analog input points that have a corresponding analog output point.

AOP1: Set Analog Output Point (AOP ID; setpoint)

Commands the analog output point specified by AOP ID to the required setpoint. This operation is intended to be used for analog output points that have no corresponding analog input point.

AIP3: Iterate to setpoint (AIP ID; setpoint, return coordinate) synchronous output: discrepancy

Commands the analog output point corresponding to the analog input point specified by AIP ID. Repeats the command until either (1) the difference between the specified setpoint and the actual scaled value read from the interface is zero or (2) the command has been attempted several times without success. In either case, the discrepancy between the specified setpoint and the scaled value obtained is returned as a synchronous output.

TAG10: Three-axis move (TAG ID; x setpoint, y setpoint, z setpoint, notification)

Commands the three axes specified by TAG ID to the specified setpoints. Returns asynchronously via notification when the device positioned according to these three axes is in the new position.

Data types

AIP ID	integer	TAG ID	integer
AOP ID	integer	x setpoint	real
scaled value	real	y setpoint	real
setpoint	real	z setpoint	real
discrepancy	real	notification	transfer vector

5. Specifying a Request Wormhole

Control is transferred from one domain to another by invocation of a wormhole process on an ADFD or SDFD. Each wormhole must be described in much the same way as any other type of process. In particular, the analyst must specify (see Figure 5.1):

- An identifier for the wormhole specification, unique within the calling domain.[1]
- The "meaning" or purpose of the wormhole, stated in terms of the calling domain. This meaning acts as the process name for the wormhole.

For each wormhole, specify also:

- Data items, if any, to be transmitted to the receiving domain. For each such data item, specify the meaning and data type.
- Similarly, specify the data items (if any) output from the wormhole when control returns to the ADFD or SDFD where the wormhole was invoked.
- Transfer vectors, if any, to be transmitted to the receiving domain.

---

[1]We choose simply to number the wormholes W1, W2, W3...Wn. Note that this does not conflict with synchronous service or process numbering: a wormhole is a distinct entity in the OOA of OOA.

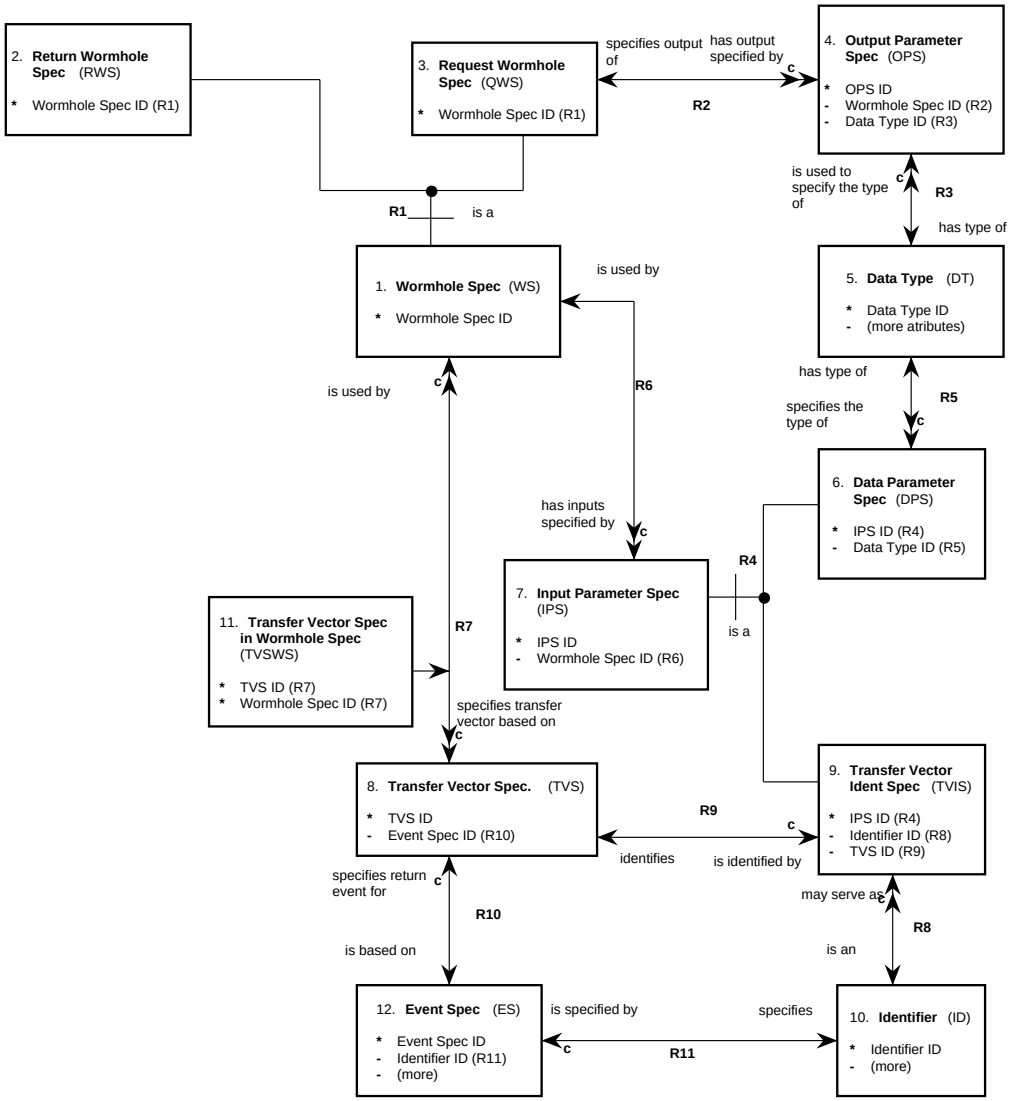


Figure 5.1: Specification of a Request Wormhole (extract from the OOA of OOA).

Observe that there is nothing in the caller's specification of a wormhole that indicates whether this wormhole will be mapped to a synchronous service or to an external event in the receiving domain. This is consistent with the goal of keeping domains well decoupled from one another. As a result, the model for the calling domain can be constructed without knowledge of precisely how the recipient domain expects to receive control.

Wormholes, like other processes, may be re-used. If a wormhole with the same identifier appears in several places in the ADFDs/SDFDs of a domain, the meaning, input and output parameters, and the types of all the parameters must be the same for all occurrences.

6. Specifying a Return Wormhole

Synchronous and asynchronous return wormholes are specified in a manner similar to request wormholes (see Figure 6.1). That is, the analyst must specify

- an identifier for the wormhole
- the purpose of the wormhole ("Return")
- data items (input to the wormhole), if any, to be transmitted to the receiving domain
- a data item, input to the wormhole, of type transfer vector or return coordinate.

A return wormhole has no outputs in the domain in which it is invoked.

Finally, note that it is not necessary to state explicitly whether the return itself is to be synchronous or asynchronous:

- If there is an input to the wormhole of type transfer vector, the return will be asynchronous.
- If there is an input of type return coordinate, the return will be synchronous

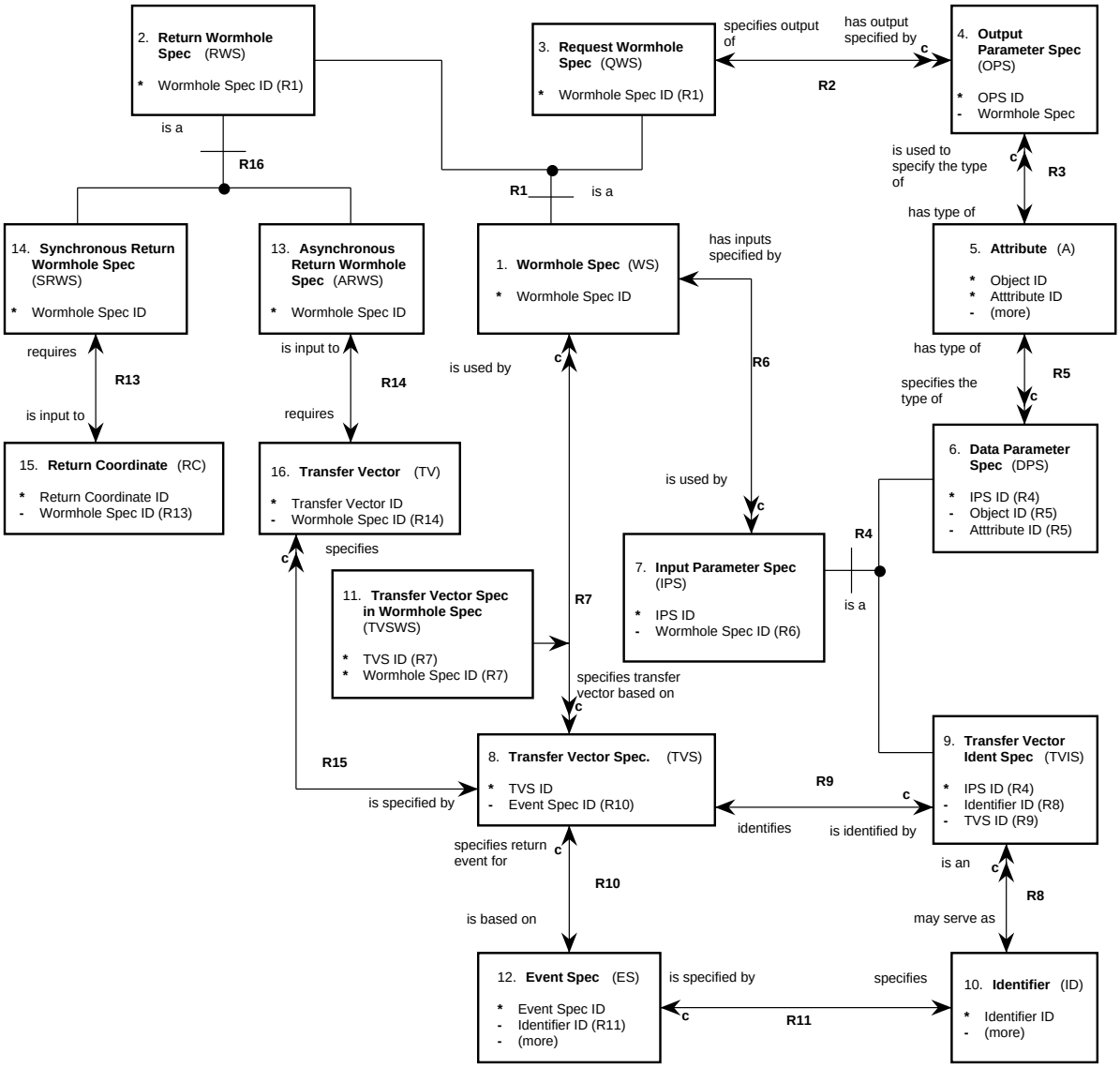


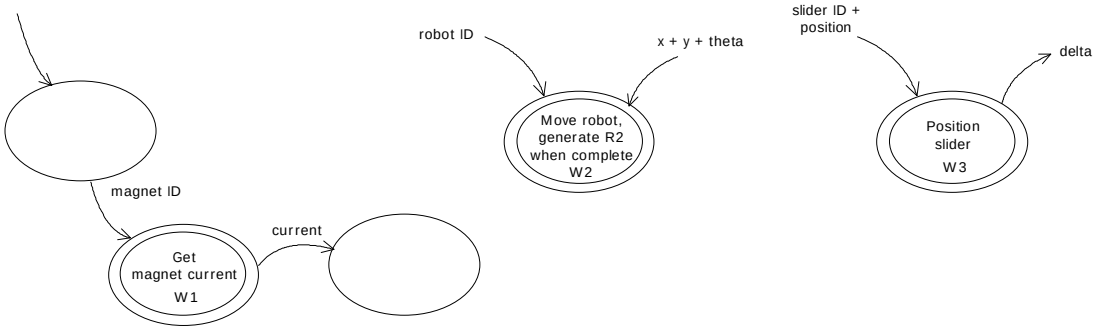
Figure 6.1: Specification of a Return Wormhole (extract from the OOA of OOA).

7. Resolving the Wormholes

Let us start with an example. In some application domain we have, among others, the following objects:

Magnet (Magnet ID, temperature, current)
Rotating-hand robot (Robot ID, x, y, θ)
Field probe (Field Probe ID, x, y, z, magnet field, Magnet ID (R))
Slider (Slider ID, position)

where the identifiers (underlined) have arbitrary values 1, 2, . . . and the remaining attributes represent the current values of various sensor-based quantities. In this application domain we also have the wormholes shown in Figure 7.1.



W1: Get magnet current. input: Magnet ID. Synchronous output: current
W2: Move Robot: input: x, y, θ , Robot ID. Generate R2: Robot move complete (Robot ID) on completion.
W3: Position slider. input: Slider ID, position. Output: delta.

Figure 7.1: Wormholes in the application domain.

The problem is to "resolve the wormholes"—that is, to construct the bridge between the application domain and the PIO domain whose interface was given in Section 4. To do this, we must:

1. Populate the participating domains.
2. Match each wormhole with an external event or synchronous service.
3. Develop correspondences between data values that cross the domain boundary.

These steps must be done prior to translation.

Populating the participating domains. Because building a bridge generally requires referring to instances in one or both domains, we must first populate the domains. Let us assume this has already been done for the application domain, and proceed to work on the PIO domain.

The PIO domain includes the following objects:

Analog Input Point (AIP ID, register, a, b, raw value, scaled value)
where scaled value = a*raw value + b
Analog Output Point (AOP ID, register, c, d, raw value, scaled value)
where raw value = c*scaled value + d
Analog Input-Output Pair (AIP ID, AOP ID)

Three Axis Group (TAG ID, AIP ID 1, AIP ID 2, AIP ID 3)

where a raw value is the value as read from the hardware interface and a scaled value is the raw value converted to engineering units (amps, °C, centimeters, etc.).

Working from signal lists supplied by the instrumentation engineers, we populate instances in the PIO domain as shown in the following somewhat abbreviated tables.

Analog Input Point				
AIP ID	**Register**	**a**	**b**	
1	1	.3	0	magnet 1 current
2	2	.1	0	magnet 1 temperature
3	3	.5	0	magnet 2 current
4	4	.1	0	magnet 2 temperature
. . .	. . .	. . .	. . .	
48	75	1	.03	slider
. . .	. . .	. . .	. . .	
96	123	10	0	robot 1 x-axis
97	124	.5	0	robot 1 y-axis
98	125	.36	-180.	robot 1 θ-axis
. . .	. . .	. . .	. . .	
114	33	1	0	field probe x-axis
115	34	1	0	field probe y-axis
116	35	1	0	field probe z axis
. . .	. . .	. . .	. . .	

Analog Output Point				
AOP ID	**Register**	**c**	**d**	
1	21	3.3	0	magnet 1 current
2	22	2	0	magnet 2 current
3	23	1.67	0	magnet 3 current
. . .	. . .	. . .	. . .	. . .
17	202	.1	0	robot 1 x-axis
18	203	2.	0	robot 1 y axis
19	203	2.78	500	robot 1 θ axis
20	143	1	0	field probe x-axis
21	144	1	0	field probe y-axis
22	146	1	0	field probe z axis
. . .	. . .	. . .	. . .	
31	13			slider position
. . .	. . .	. . .	. . .	. . .

Analog Input Output Pair		
AIP ID	**AOP ID**	
1	1	magnet 1 current
3	2	magnet 2 current
5	3	magnet 3 current
. . .	. . .	
48	31	slider
. . .	. . .	
96	17	robot 1 x axis
97	18	robot 1 y axis
98	19	robot 1 θ axis
. . .	. . .	
114	20	field probe x axis
115	21	field probe y axis
116	22	field probe z axis
. . .	. . .	

Three Axis Group				
TAG ID	**AIP ID 1**	**AIP ID 2**	**AIP ID 3**	
1	96	97	98	robot 1 x, y, θ
2	114	115	116	field probe x, y, z
. . .	. . .	. . .	. . .	

Matching wormholes with control reception points. For each request wormhole in one domain (here, the application domain), we must identify a synchronous service or external event in another domain (PIO in this example). A number of factors must be taken into account when making such a match.

1. Functionality. The functionality required by the request wormhole must be the same as the functionality supplied by the control reception point[2]. The only difference permitted is a semantic shift: the vocabulary used to describe the functionality is expected to be different in the two domains.

2. Inputs. Each data item input to the wormhole must correspond to an input to the control reception point (CRP). The corresponding inputs must be of compatible data types.

   Special case: If a CRP shows a transfer vector as input, then the wormhole must show input data items that serve as identifiers for the event to be generated on the basis of the transfer vector. The transfer vector is considered to correspond to all data items that constitute the required identifier.

3. Synchronous outputs. Each synchronous output (if any) from the wormhole must correspond to a synchronous output from the CRP. The corresponding synchronous outputs must be of compatible data types.

4. Asynchronous returns and outputs. If the specification of the wormhole requires that an event be returned to the requesting domain, then the CRP must show a transfer vector as an input data item. If the event carries supplemental data items, the CRP must supply such data items as output with the transfer vector. These data items must be compatible in type with the supplemental data items defined in the requesting domain.

In order to identify the CRP that corresponds to each wormhole, it is helpful to bring the definitions of all request wormholes from one domain together with all CRPs of the other domain. One way of doing this is shown in the following table:

Domain	**Ident**	**Meaning**	**Inputs**	**Synch outputs**	**Asynch return**	**Asynch outputs**
Application	W1	Get magnet current	Magnet ID	current	–	NA
Application	W2	Move robot	x, y, θ, Robot ID	–	R2: (Robot ID)	none
Application	W3	Position slider	Slider ID, position	delta	–	NA
PIO	S1	Read analog input point	AIP ID, return coordinate	scaled value	–	NA
PIO	AIOP1	Set analog input point	AIP ID, setpoint	–	–	NA
PIO	AOP1	Set analog output point	AOP ID, setpoint	–	–	NA
PIO	AIP3	Iterate to setpoint	AIP ID, setpoint, return coordinate	discrepancy	–	NA
PIO	TAG10	Three-axis move	TAG ID, x setpoint, y setpoint, z setpoint, notification	–	notification	none

[2]For economy of expression, we refer hereafter to a synchronous service or external event as a "control reception point", or CRP.

After considering the functionality of both the wormholes and the CRPs, we make the following correspondences:

W1:	Get magnet current	→	S1: Read analog input point
	Magnet ID	→	AIP ID
	current	→	scaled value

W2:	Move robot	→	TAG10: Three-axis move
	Robot ID	→	TAG ID
	Robot ID + R2	→	notification
	x	→	x setpoint
	y	→	y setpoint
	θ	→	z setpoint

W3:	Position slider	→	AIP3: Iterate to setpoint
	Slider ID	→	AIP ID
	position	→	setpoint
	delta	→	discrepancy

Many of these correspondences develop during execution: For example, in the case of W1→S1, the PIO domain develops a scaled value and returns it to the application domain, which interprets the value as a magnet current.

Exactly how the correspondence Robot ID+R2→transfer vector is developed is a responsibility of the architecture. It is likely that the R2 part of the correspondence will be developed at translation time, with the transfer vector being finally completed at run time using the value of Robot ID that was passed to PIO. Note that we cannot say more in the general case because the answer depends on exactly how the architecture implements the concept of an event.

What if there is no match for a wormhole? We normally think of a client domain as imposing requirements on its servers. From this perspective, a wormhole in a client domain makes explicit a number of very detailed requirements for the server, and we would expect these requirements to be reflected faithfully in the server's Domain Interface document. In this situation you will always be able to find a CRP that matches each wormhole exactly.

However, in the everyday world of software development, work often proceeds out of order with respect to such an idealized plan: You may have an existing server domain all ready for reuse, you may have purchased a domain, or—for whatever reason—your project may have decided to produce some server domains before the clients. In such a case you may not be able to find a CRP that is a perfect match for a wormhole, so remember that the conventional concept of negotiated interfaces still applies. Depending on the particulars of the situation, you may choose to modify either the client or the server models to achieve a match. This should not be seen as a serious problem. In our experience, putting a domain interface to a practical, realistic test—from either the client or the server side—only results in stronger, more knowledgeably-conceived models.

Matching data values. Now that we have matched the wormholes with CRPs and the input and output parameters from the application domain with the corresponding parameters in the PIO domain, there is still one more step: matching the data values that cross the domain boundary. Because the PIO domain has taken on the responsibility of converting the data values at the hardware interface to engineering unit values as needed by the application, the work that remains is entirely straightforward. We need only

prescribe how identifiers in the application domain are mapped to identifiers in PIO. This can be done by means of the following bridge table[3]:

Application Domain					***PIO Domain***		
object name[4]	identifier attribute name	identifier attribute value	non-identifying attribute name		object name[4]	identifier attribute name	identifier attribute value
Magnet	Magnet ID	1	current		AIP	AIP ID	1
Magnet	Magnet ID	1	current		AOP	AOP ID	1
Magnet	Magnet ID	1	temperature		AIP	AIP ID	2
Magnet	Magnet ID	2	current		AIP	AIP ID	3
Magnet	Magnet ID	2	current		AOP	AOP ID	2
Magnet	Magnet ID	2	temperature		AIP	AIP ID	4
Magnet	Magnet ID	3	current		AIP	AIP ID	5
Magnet	Magnet ID	3	current		AOP	AOP ID	3
Magnet	Magnet ID	3	temperature		AIP	AIP ID	6
. . .	. . .	. . .	. . .		. . .	. . .	. . .
RH Robot	Robot ID	1	x		AIP	AIP ID	96
RH Robot	Robot ID	1	x		AOP	AOP ID	17
RH Robot	Robot ID	1	y		AIP	AIP ID	97
RH Robot	Robot ID	1	y		AOP	AOP ID	18
RH Robot	Robot ID	1	θ		AIP	AIP ID	98
RH Robot	Robot ID	1	θ		AOP	AOP ID	19
. . .	. . .	. . .	. . .		. . .	. . .	. . .
Field probe	Field probe ID	1	x		AIP	AIP ID	114
Field probe	Field probe ID	1	x		AOP	AOP ID	20
Field probe	Field probe ID	1	y		AIP	AIP ID	115
Field probe	Field probe ID	1	y		AOP	AOP ID	21
Field probe	Field probe ID	1	z		AIP	AIP ID	116
Field probe	Field probe ID	1	z		AOP	AOP ID	21
. . .	. . .	. . .	. . .		. . .	. . .	. . .
Slider	Slider ID	1	position		AIP	AIP ID	48
Slider	Slider ID	1	position		AOP	AOP ID	31
. . .	. . .	. . .	. . .		. . .	. . .	. . .

The bridge table constitutes the final link between the two domains. Note that there is nothing in the structure of this table that is particular to either of the participating domains: all of the connection information has been captured in data only. As a result, the code that processes this table—the bridge code—is indeed generic and independent of either domain. This is exactly the property we posited when describing a bridge as a clear pane of glass.

Acknowledgements

As we proceeded through the formalization of Recursive Design, no topic provoked more discussion among our clients and colleagues than that of bridges and wormholes. We are grateful to all who participated in these sessions, and in particular, to Kent Mingus, Gregory Rochford, and John Wolfe for their exploration of alternatives, precision of thought, and downright stamina.

[3]The structure of this table—and of bridge tables in general—is discussed in detail in Chapter <x>.
[4]Some object names have been abbreviated to fit into the table.