Aspects and Multi-aspects

6

So far we have encountered different aspects that are present in the same System Entity Structure. However, these aspects were under *different* entities so that no entity had more than one aspect. When an entity has only one aspect, this allows you to decompose the model it represents in one way (as well as provide couplings for the associated composition). In the next section, you will see how there can be different aspects for the *same* entity and how this allows you to decompose a system in different ways. After that, we will discuss the concept of multi-aspect which provides a uniform way to associate an unlimited number of related aspects with the same entity. Each multi-aspect effectively opens up a large space of simulation models with an unbounded variety of possibilities for coupling their components. So, finally, we discuss how coupling of components can be specified for such multi-aspects in uniform ways that greatly reduce the amount of data entry required.

6.1 Multiple Aspects (Decompositions)

6.1.1 Expressing Different Aspects for Same Entity

Consider adding the following to the SES for MSProcessSystem (Chap. 3).

```
From the fastProcess perspective, MSProcessSystem is made of
User,ClarifyObjectivesPhase, getModelFromRepository,
ExecuteModelPhase, and InterpretResultsPhase!
```

Here we add a second aspect to the SES which has a different label, “fastProcess”, to distinguish it from the first which was labeled “process”. The newly added aspect represents a decomposition of the modeling and simulation process in which the gathering data phase is skipped and, rather than being constructed, a model is retrieved directly from a repository. This foreshortening of the process is consistent with the label “fastProcess” and would be suitable when the M&S development process is constrained by time and resource availability. Because we are skipping the data gathering phase, the output of the clarifying objectives phase now has to drive the model retrieval. This is implemented in the coupling:

B.P. Zeigler, H.S. Sarjoughian, *Guide to Modeling and Simulation of Systems of Systems*, Simulation Foundations, Methods and Applications, DOI 10.1007/978-0-85729-865-2_6,

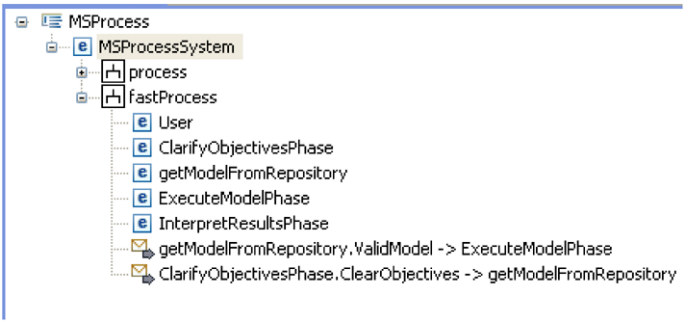

Fig. 6.1 Outline of the SES for MSProcessSystem after fastProcess addition

```
From the fastProcess perspective, ClarifyObjectivesPhase sends
ClearObjectives to getModelFromRepository!
```

Also the output of the model must be sent to start the model execution:

```
From the fastProcess perspective, getModelFromRepository sends
ValidModel to ExecuteModelPhase!
```

After adding the fastProcess aspect and the above couplings to the SES, its outline appears in Fig. 6.1. Note that there are now two aspects under the MSProcessSystem entity. To produce a specific DEVS model, eventually one of these aspects must be selected.

Exercise

Fill in the rest of the couplings for this decomposition in the SES for the M&S process. Use the MS4 Me tool to generate the outline and compare with Fig. 6.1.

6.1.2 Pruning of Aspects

Recall that an aspect represents a decomposition and that at most one decomposition per entity can give rise to a simulation model. Therefore, you must select a single aspect whenever there are more than one aspects under an entity. Returning to the pruning interface of MS4 Me with the amended SES, you will now see MSProcessSystem included in the list of prunable items on the left hand side. Highlighting that entity will now allow you to select one of its two aspects. Since aspects combine with specializations, each selection of an aspect will enable pruning of all items in the substructure (specializations or multiple aspects under entities). Accordingly, there are now two families of pruned entity structures for the M&S Process SES—one for the original process aspect and one for the fastProcess aspect.

Exercise

Using the MS4 Me interface, create at least two completely pruned entity structures stemming from the selection of the aspects under MSProcessSystem.

6.1.3 Aspects: Perspectives and Abstractions

As the name implies, the SES concept of aspect allows you to represent taking different perspectives on the same real world system or process and including them within the same structure. The great benefit of viewing the same thing from different perspectives is that this may well lead to simplifications or idealizations that allow dispensing with a lot of the complexity inherent in the real world. System abstractions derived from different perspectives often can be treated in a stand-alone or quasi-independent manner. Unfortunately, although an effective abstraction enables you to get into the right ballpark from that particular point of view, its assumptions tend to conflict with those of other abstractions when pushed beyond its limits. The advantage of including multiple aspects in the same structure (a single SES) is that we can work with them all together as a whole when we need to.

Consider for example, modeling multi-person games on the Internet. The Internet can foster the illusion of a virtual space such that players may focus on the rules of the game no matter where they are geographically located in the real world. Thus, modeling such a game from the player's perspective can represent the constraints of the game and the players' strategies for dealing with them. From the game host's point of view however, supporting a menu of several games does not critically depend on a particular game but only on their general characteristics that relate to providing the speed of response necessary to maintain the illusion that players are not separated by distances between them. Let's recognize that these different abstractions are driven by different objectives—the player's goals of working out winning strategies and the host's objectives in providing the virtual world's illusion. And while initially self-contained, they may interact at important junctures. For example, a player may realize that he enjoys a slight speed advantage and capitalize on it in making a play. Likewise, the game developer may realize that a particular game may stress the network more than average and so require special treatment. A third aspect which includes the combined entities in both aspects can be added to take account of the interactions that arise under circumstances where the original assumptions fail.

Exercise

Develop an SES for the game/network scenario that has three aspects under its root entity. In the game aspect, players A and B vie to be the first to recognize an object that appears (virtually) simultaneously on their separate screens. In this abstraction there are the players, their screens, and the game serving computer. The focus is on the pattern recognition skills of the players and no Internet representation is present. In the second aspect, the game server and players' computers are modeled as nodes on the Internet with a representation of the latencies experienced by packets sent from node to node. The goal is to assure that despite such latencies, the game software assures the virtual simultaneity of screen displays. The third aspect allows players to perceive breakdowns in the simultaneity of screen displays and includes players to exploit such perceptions.

Exercise

You can obtain some descriptions of building architectures drawn from Wikipedia http://en.wikipedia.org/wiki/Architectural_style as well as particular house styles and building materials http://en.wikipedia.org/wiki/Townhouses. Develop an SES for a house that will allow you to design a particular house by pruning. The SES should be such that you could prune it to obtain any one of several house styles. For example, from the sentence, "Townhouses usually consist of multiple floors and have their own outside door as opposed to having only one level and an interior hallway access", you can infer that different aspects might exist for single level and multilevel styles and that outside door is an option.

6.2 Multi-aspects—Multiple Related Decompositions of an Entity

6.2.1 Limitations of Aspects

So far you have learned that you can express several decompositions of an entity using one aspect for each decomposition. However, there is a limitation that we now address. This limitation arises from the fact we need to provide a finite list of entities for each aspect. This is awkward when we want to let an aspect have an indefinite number of entities. For example,

```
From the top perspective, the People are made of Alice, Bill,
and Charlie!
```

states that the entity, People, is composed of specifically three entities with names, Alice, Bill, and Charlie.

To increase the number of entities to four, we need to explicitly add a fourth named entity, say David. This means changing the SES, replacing the above sentence by this one:

```
From the top perspective, the People are made of Alice, Bill,
Charlie, and David!
```

Alternatively, we can use place both aspects with different names in the same SES:

```
From the topThree perspective, the People are made of Alice,
Bill, and Charlie!

From the topFour perspective, the People are made of Alice,
Bill, Charlie, and David!
```

Although, this approach allows us to select aspects of the same entity with different numbers of entities, we have to list the decompositions explicitly. Besides being tedious and error prone, the choice is always limited by the finite set of aspects actually defined.

6.2.2 Multi-aspect Restructuring

The concept of *multi-aspect* is intended to address the limitation of having to explicitly declare decompositions having different numbers of entities. To illustrate, consider the following pair of statements:

```
From the top perspective, the World is made of People and
Environment!
From the multiPerson perspective, People are made of more than
one Person!
```

The second statement generates a multi-aspect in the SES. A multi-aspect is a special case of an aspect except that it has only one child called its *multi-entity*—which can generate a specified number of copies (Zeigler and Hammonds 2007). In the above example, People is the entity that has the multi-aspect labeled by multiPerson and Person is its multi-entity of multiPerson.

To *expand* a multi-entity (i.e., to generate copies) you add a specialization to it, for example:

```
Person can be id in index!
```

Then in a pruning file you include two statements such as:

```
restructure multiaspects using index!

set multiplicity of index as [3]for Person!
```

where the first statement tells the pruner to expand using the specialization labeled by index and the second tells how many copies to make. The specialization entity, id is used to identify the copies – for example, id0_Person, id1_Person, id2_Person, are generated. These new entities are added to the (ordinary) aspect under the grandparent entity and the mult-aspect is removed from the SES. For example, the new SES after the given restructuring can be described by

```
From the top perspective, the World is made of Environment,
id0_Person, id1_Person and id2_Person!
```

Figure 6.2 illustrates this restructuring where the original SES is depicted in Fig. 6.2(a) and the restructured SES is depicted in Fig. 6.2(b). Note that the multi-aspect multiPerson has been eliminated and the copies of the multi-entity Person are added to the aspect under World. Note that originally, Person was the grandchild of World and now the copies of Person are the children of World.

6.2.3 Pruning Multi-aspects

The multi-entity of a multi-aspect can have a sub-SES below it. For example, we might have

```
Person can be short, medium, or tall in height!
```

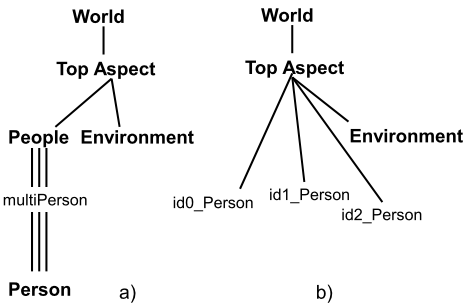

Fig. 6.2 Multi-aspect restructuring

Here each of the copies inherits this sub-SES and during pruning each these copies can be pruned independently. This can be visualized as equivalent to replacing the specialization for Person by the same specialization added to each new entity:

```
id0_Person can be short, medium, or tall in height!

id1_Person can be short, medium, or tall in height!

id2_Person can be short, medium, or tall in height!
```

The pruning file can then specify pruning selections for none, some or all the entities. For example,

```
select medium from height for id1_Person!
```

selects medium height for the person with identity id1. As general is the case, selections not made explicitly in the pruning file are made randomly.

6.2.4 Multi-aspect Uniform Coupling

After restructuring, the entities of a multi-aspect can be coupled by referring to them by name. For example,

```
From the top perspective, id0_Person sends Bye to id1_Person!
```

creates a coupling from outBye of id0_Person to inBye of id1_Person. Such individualized coupling can be tedious and error prone as the numbers of entities increases. Therefore, we need the capability to create couplings that can be uniformly stated similarly to the logic quantifier, *for all*. For example,

```
From the top perspective, World is made of People and
Environment!
From the multiPerson perspective, People are made of more than
one Person!
Person can be id in index!
From the top perspective, all Person sends Bye to id0_Person!
From the top perspective, all Person sends Hello to all Person!
```

This creates the coupling pattern shown in the simulation viewer capture of Fig. 6.3. Here every Person's output port outBye is coupled to id0_Person's input port inBye.

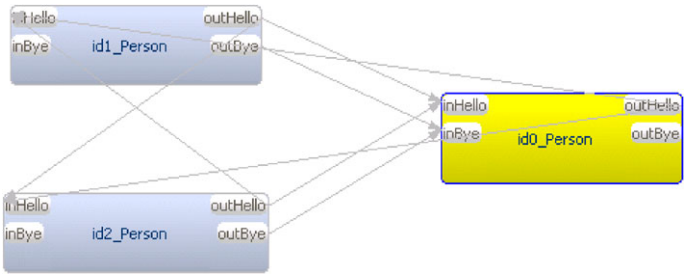

Fig. 6.3 Illustrating all-to-all coupling pattern

Every Person's output port outHello is coupled to every Person's input port inHello (the exception that there is no self-coupling i.e., no output port of a component is coupled to any of its input ports.)

Uniform coupling can be employed to couple entities between different restructured multi-aspects. For example, in the following SES, all boys say Hello to each other, all girls say Hello to each other, and all girls greet all boys.

```
From the dating perspective, the DatingGame is made of Boys
and Girls!
From the boy perspective, Boys are made of more than one Boy!
Boy can be shortish, medium, or tall in height!

From the girl perspective, Girls are made of more than one Girl!
Girl can be shortish, medium, or tall in height!

From the dating perspective, all Boy sends Hello to all Boy!
From the dating perspective, all Girl sends Hello to all Girl!
From the dating perspective, all Girl sends NiceToMeetYou to
all Boy!
```

Here note that restructuring is based on the height specialization. This will work in a straight forward manner provided you don't use height as the basis for identifying multi-entity copies (in other words, "set multiplicity of height ..." does not appear in the pruning file).

We introduce another form of uniform coupling with the *each-to-each* construct shown in the following:

```
From the dating perspective, each Boy sends Invitation to each
Girl!
```

As illustrated in Fig. 6.4, this results in each boy having its output port outInvitation coupled to the inInvitation port a girl having the same height. Matching is done on the basis of having the same prefix in the pruned name of an entity (e.g., tall in tall_boy). Couplings are not added for matches that cannot be made—for example, if the numbers of boys and girls are not equal. Other coupling types, such as all-to-each, are supported.

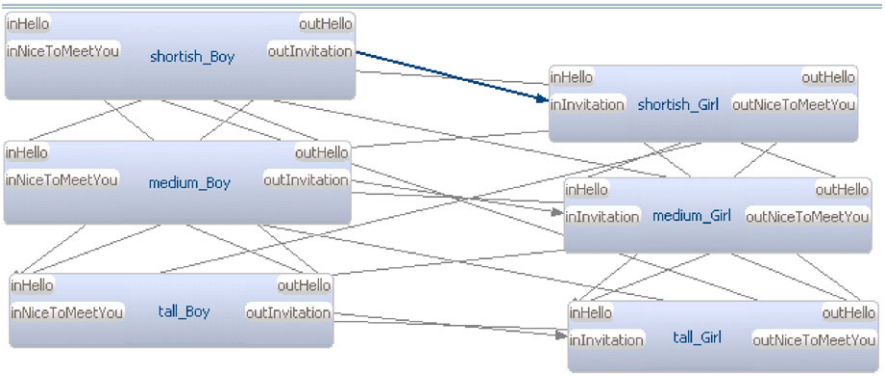

Fig. 6.4 Coupling based on matching

6.2.5 One-to-All and All-to-One Coupling

Coupling any entity to the entities of a multi-aspect can be specified in using the one-to-all form as in:

```
From the dating perspective, short_Boy sends Greeting to all
Girl!
```

This will work provided that the entity to be coupled is in the same coupled model as the multi-entities. For example, if MatchMaker were added to the Dating Game, then one could specify:

```
From the dating perspective, MatchMaker sends MakeInvitation
to all Boy!
```

The source entity can also be the coupled model that contains the multi-entities, so that we can specify:

```
From the dating perspective, DatingGame sends MakeInvitation
to all Boy!
```

All-to-one coupling works in a predictably similar manner to one-to-all coupling. For example,

```
From the dating perspective, all Boy sends Preference to
MatchMaker!
```

Exercise

Extend the following SES text so that every person sends a Sit instruction to his/her own dog, and every dog can Bark so that it is heard by the whole world. Write a pruning file so that there are 5 people and 3 Dogs.

```
From the top perspective, World is made of People and Dogs!
From the person perspective, People are made of more than one
Person!
From the dog perspective, Dogs are made of more than one Dog!
```

```
Person can be id in index!
Dog can be id in index!
```

Exercise

A proposed satellite system architecture contains relay satellites and image-taking satellites. Imaging requests from a single ground station are sent to relay satellites which can pass on the requests to all imaging satellites. After taking an image, an imaging satellite can send it to all relay satellites which can relay it back to the ground station. Develop an SES with multi-aspects for each type of satellite that employs one-to-all, all-to-all, and all-to-one coupling. Write a pruning file that specifies 2 relay and 10 imaging satellites (See Chap. 13).
More on "all" and "each" coupling appears in Chap. 9 in the context of DEVS distributed simulation and in Chap. 15 on Cloud systems.

6.2.6 Hierarchical Construction with Multi-aspects

Multi-aspect restructuring, pruning, and coupling paradigms apply to hierarchical construction as well. For example, when a multi-aspect is expanded into entities, each entity may itself have a multi-aspect in need of expansion. This leads to hierarchical construction where the components at one level are constructed from components at the next lower level. A global situation awareness database system consisting of multiple geographically distributed enclaves offers an example. Each enclave contains a database server that serves any number of clients. There is no direct connection among clients. Database servers are connected to each other in an all-to-all fashion so that a client's update in one enclave is propagated to all the others.

To express this system in MS4 Me we create multi-aspects for multiple enclaves and for multiple clients within each enclave which leads to an SES description:

```
From the overall perspective, MultiEnclaveNet is made of
Enclaves!
From the multiEnclave perspective, Enclaves are made of more than
one Enclave!
//sharing of updates among enclaves
From the overall perspective, all Enclave sends DBUpdate to
all Enclave!
Enclave can be enclaveid in index!

From the enclave perspective, Enclave is made of DBServer and
Clients!
From the multclient perspective, Clients are made of more than
one Client!
Client can be clientid in index!
//interaction of clients and servers within an enclave
From the enclave perspective, all Client sends DBQuery to
DBServer!
From the enclave perspective, all Client sends DBUpdate to
DBServer!
From the enclave perspective, DBServer sends DBResponse to all
```

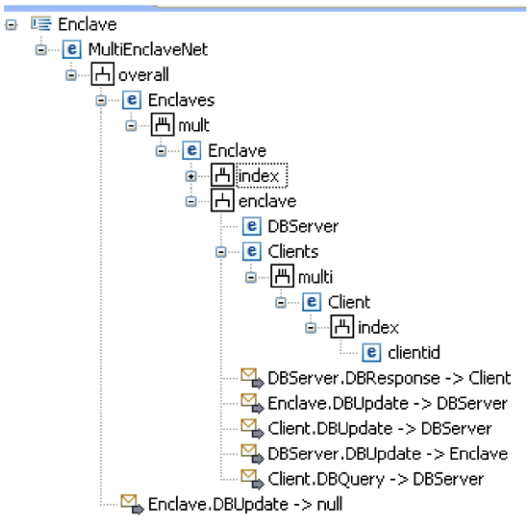

Fig. 6.5 Outline for MultiEnclaveNet

```
Client!
//connecting the server within an enclave to the enclosing
enclave
From the enclave perspective, DBServer sends DBUpdate to
Enclave!
From the enclave perspective, Enclave sends DBUpdate to
DBServer!
```

The outline in Fig. 6.5 shows the outline for the MultiEnclaveNet SES. After pruning and transformation, MultiEnclaveNet will become a coupled model that contains one or more Enclaves, each of which becomes a coupled model that contains a DBServer and multiple clients.

The numbers of Enclaves and Clients can be specified in a pruning file containing statements such as:

```
restructure multi-aspects using index!
set multiplicity of index as [2] for Enclave!
set multiplicity of index as [3] for Client!
```

Note that it is permissible, and desirable, to use the same specialization (here index) for different multi-aspects in the SES, where the multiplicity can be differentially specified in the pruning file. If different names are used, e.g., enclaveIndex and clientIndex, then a bottom up order must be given for restructuring, as in

```
restructure multi-aspects using clientIndex then enclaveIndex!
```

Note that the number of clients in each enclave is the same since we can't specify them individually. This is a limitation on the expressiveness of the current version of MS4 Me which can be removed by supporting specification of multiplicity using context.

Exercise

Using MS4 Me enter the above SES and test it using the animation feature. Try specifying different numbers of enclaves and clients per enclave. How might it become possible to specify different numbers of clients in each enclave?

Exercise

In the space system in Exercise xx, the imaging and relay satellites are composed of modules where some of the modules are different for each type of satellite. For example, imaging satellites have cameras whereas relay satellites have more powerful transceivers able to communicate with the earth. Elaborate on your SES for Exercise xx by attaching a multi-aspect for modules under the multi-entity of the relay as well as the imaging satellite.

Exercise

A relation is a set of pairs, where each pair has a left element and a right element. Here, the left element belongs to a domain set and the right element belongs to a range set. This structure can be generated by an SES described by:

```
From the rel perspective, Relation is made of pairs!
From the mult perspective, pairs is made of more than one pair!
pair can be id in index!
From the pr perspective, pair is made of left and right!
left can be key in domain!
right can be value in range!
```

To specify a particular relation, you can write a pruning file starting with the following (as an example):

```
restructure multi-aspects using index!
set multiplicity of index as [4] for pair!
set multiplicity of domain as [6] for left!
//creates elements key0, key1, ...., key5 for selection from
domain
set multiplicity of range as [3] for right!
```

Using context sensitive pruning (Chap. 8), add a selection statements to define a particular relation compatible with the above statements. Hint: to specify a pair, (key5, value0) for the first pair, write:

```
select key5 from domain for left under id0_pair!
select value0 from range for right under id0_pair!
```

6.2.7 Uniform Pairwise Coupling

So far we have used multi-aspects to specify compositions with components that are derived from the same entity and whose particulars are decided while pruning. We have also been able to specify couplings among these components that are uniform

in the sense of applying to all in the same manner (see Sect. 6.2.3). However, there are important kinds of coupling patterns that don't fit this mold. For example, cellular automata connect up their components in a mesh pattern based on a geometrical coordinate grid (Wainer et al. 2010; Muzy and Hu 2008). Tree automata lay out their components on a tree structure where starting from the root, each non-leaf node has, say two, children and there are no couplings other than between parents and children. Indeed, there are numerous such connection schemes, and variations thereof, of possible interest to modelers. To bring each of these schemes within the framework of multi-aspects presents a challenge since it is not possible to specify them all in advance. Instead, we can provide a mechanism that allows modelers to specify their own patterns.

To explain this approach, we can envision a coupling scheme which consists of two parts: (1) specification of network connectivity, and (2) the coupling of ports between connected components. The network connectivity can be described by a mathematical graph of elements called nodes, and pairs of nodes, called edges. Furthermore, in our context, the graph is a directed graph, or digraph, meaning that the pairs are ordered. In an ordered pair, the left hand node can be called the sender and right hand node is the receiver. The nodes of the graph are in one-one correspondence with the components and the form of the graph is given by specifying a subset of edges. Such a subset can be regarded as being selected from all the available ordered pairs using a pruning process. Each pair corresponds to a direct connection between associated components. In summary, a directed graph sets up the pair-wise connections for the network connectivity.

This leaves the port-to-port coupling to be specified for each connection in the directed graph of a coupling scheme. Recall that in general different pairs of components can have different port-to-port couplings. However, to simplify the coupling specification and in the spirit of uniformity, we will force the port-to-port coupling to be the same across all such pairs.

To see how this *uniform coupling* rule works, consider the coupling example in Sect. 6.2.3,

```
From the top perspective, id0_Person sends Bye to id1_Person!
```

This creates a coupling from outBye of id0_Person to inBye of id1_Person. Here the ordered pair of components is (id0_Person, id1_Person), or for short (id0,id1), and the port-to-port coupling is (outBye, inBye). The uniform coupling requirement dictates that all edges in the network digraph have the same port-to-port coupling. So for example, if (id2,id3) is a connected pair, then a port-to-port coupling (outBye, inBye) is assigned to this pair. The corresponding natural language statement would be:

```
From the top perspective, id2_Person sends Bye to id3_Person!
```

To make the uniform coupling rule operational, we need a way to specify network connectivity and then a way to specify port-to-port coupling. For network connectivity, the question is: How can you select a set of pairs of nodes to form a directed graph? To answer this question, suppose the following appears in an SES file:

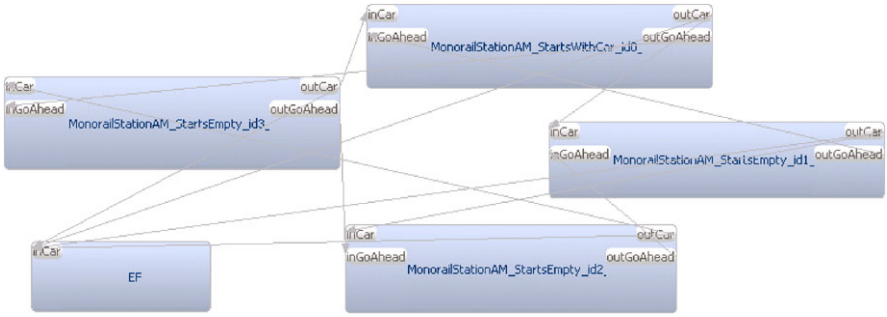

Fig. 6.6 Monorail composition

```
From the topsys perspective, MonorailSystem is made of EF,
MonorailStations, and circleCouplingSpecification!

From the multiStation perspective, MonorailStations is made of
more than one MonorailStation!

MonorailStation can be id in index!
```

Note the presence of a special entity, circleCouplingSpecification that has been added as a sibling to MonorailStations which contains the multiStation multi-aspect. When the pruner interprets this file, the suffix, CouplingSpecification, will be recognized and the pruner will expect to see further detail in the pruning file The prefix, circle, is the name of the set of pairs intended to form the digraph for the network of MonorailStations. To explicitly specify such a set of pairs, you can write, for example:

```
add coupling circlepair from id0 to id1 in index!
add coupling circlepair from id1 to id2 in index!
add coupling circlepair from id2 to id3 in index!
add coupling circlepair from id3 to id0 in index!
```

which creates the pairs (id0,id1), (id1,id2), (id2,id3), (id3,id0) forming a cycle in the graph with nodes id0,..,id3.

Next you can attach port-to-port couplings to these pairs. For example, to associate couplings with the circleCouplingSpecification you can write

```
for circle leftnode sends Car to rightnode!
for circle rightnode sends GoAhead to leftnode!
```

where we note that since the pairs are ordered, there is a node on the left (called leftnode), and a node on the right (called rightnode) . Furthermore, by the uniform coupling rule, the first statement specifies that each left node sends Car to its paired right node, and each right node sends GoAhead to its paired left node.

Viewed in the simulation viewer (Fig. 6.6) we can see the network connectivity cycle digraph together with the port-to-port coupling specified by the pairs.

Finally, to provide the right context for all the information needed in the pruning file to allow such couplings to be properly interpreted, we add the statement:

```
write coupling specification for MonorailStation and index
based on circle!
```

This identifies the name of the multi-entity in the multi-aspect (MonorailStation), and the specialization (index) to be used in the coupling specification named by the circle.

6.2.8 Predefined Coupling Specifications

At this point, we see that separating the node-to-node network connectivity from the port-to-port coupling specification using uniform coupling rule greatly reduces the number of items to be specified because you avoid having to separately give the couplings for each node pair. Never-the-less you still have to explicitly list all the node pairs to specify the network connectivity and this part still can be tedious and error-prone. To provide further support, MS4 Me provides some predefined forms of network digraphs that you can put to use. For example, replacing the general "write coupling specification" just mentioned with the following statement in the pruning file will create a cycle of nodes to be associated with the circleCouplingSpecification:

```
write cyclic specification for MonorailStation and index based
on~circle!
```

Such a specification effectively selects a subset pairs forming a cycle of size specified in the multiplicity statement. So in the end, instead of explicitly listing all couplings, you can use the following compact set of statements in the pruning file:

```
restructure multi-aspects using index!
set multiplicity of index as [4] for MonorailStation!
write cyclic specification for MonorailStation and index based
on~circle!
for circle leftnode sends Car to rightnode!
for circle rightnode sends GoAhead to leftnode!
```

This file will create the same coupled model illustrated in Fig. 6.6.

Cellular and tree specifications are examples of other forms of predefined coupling specifications available for use in MS4 Me. Cellular specification offers an example of multidimensional space expansion for a multi-aspect. Consider the SES:

```
From the cell perspective, cellspace is made of cells,
cellEWCouplingSpecification, and cellNSCouplingSpecification!
From the mult perspective, cells are made of more than one cell !
cell can be x or y in location!
```

A pruning script to create a standard Moore neightborhood (immediate North, South, East, West neighbors) is:

```
restructure multiaspects using location !
set multiplicity of location as [10,10] for cell !
```

```
for cellEW leftnode sends East to rightnode!
for cellEW rightnode sends West to leftnode!
write cellular specification for cell and location based on
cellEW!

for cellNS leftnode sends South to rightnode!
for cellNS rightnode sends North to leftnode!
write cellular specification for cell and location based on
cellNS!
```

Exercise

Write an atomic model to provide the behavior of a monorail station that sends a car to the next station in a cycle but only after it has received a GoAhead from that station indicating that no car currently occupies it (the car it previously hosted has moved ahead.) A station must wait until a the car it sends to the next station actually reaches there before sending a GoAhead to the previous station (See Hwang and Zeigler (2009) for a representation and analysis of a Monorail system in FDDEVS.)

Exercise

In the previous exercise, instead of the atomic model, write a coupled model for a monorail station that explicitly decomposes the station into a platform and the track to the next station. This will allow a station to send a GoAhead to the previous station as soon as the car it is releasing is on the track to the next station.

Exercise

Add a specialization that allows a choice between the atomic and coupled models you have constructed.

Exercise

Add another specialization that allows pruning each station to start empty or with a car.

6.3 Summary

This chapter started with a discussion of how different aspects can be associated with the same entity and how this allows you to decompose a system in different ways. This led to a consideration of the concept of multi-aspect which provides a uniform way to associate an unlimited number of related aspects with the same entity. Pruning a multi-aspect involves setting its multiplicity and restructuring it into an ordinary aspect with the specified number of components. We saw that pruning of multi-aspects effectively open up a large space of simulation models with an unbounded variety of possibilities for coupling their components. Unfortunately, unless properly managed, this variety can also entail enormous amounts of detailed data entry which can be tedious and error prone. This led to development of a uniform coupling rule which separates node-to-node network connectivity (specified

by a directed graph) and port-to-port coupling which is forced to be uniform across all network connections. Some commonly employed schemes such as cyclic, cellular, and tree compositions have well defined digraphs with uniform couplings so they fit this mold.

References

Hwang, M. H., & Zeigler, B. P. (2009). Reachability graph of finite & deterministic DEVS networks. *IEEE Transactions on Automation Science and Engineering*, *6*(3), 454–476.

Muzy, A., & Hu, X. (2008). Specification of dynamic structure cellular automata & agents. In *Proc. of the 14th IEEE Mediterranean electrotechnical conference, MELECON2008* (pp. 240–246).

Wainer, G., Liu, Q., Dalle, O., & Zeigler, B. P. (2010). Applying cellular automata and DEVS methodologies to digital games: a survey. *Simulation & Gaming*, *41*(6), 796–823.

Zeigler, B. P., & Hammonds, P. (2007). *Modeling simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. Boston: Academic Press, 448 pages.