GReAT Tutorial

Table of Contents

1. Introduction

This tutorial describes the steps necessary to set up the House2Order transformation from scratch. Please note that the House2Order transformation is readily available in the GReAT package. To open the transformation, select

Start Menu/Programs/GReAT/Samples/House2Order

and click on House2Order. This will invoke GME, and the transformation will open automatically. You can browse through the transformation to get familiar with the structure and building blocks of GReAT, and/or you can execute the transformation by a few mouse clicks (see section 5 or 6 for details on how to execute GReAT transformations). This tutorial will guide you through the GReAT development process by providing a step-by-step guide to create the House2Order transformation . At the end of this tutorial you will be able to create any GReAT transformation that has complexity similar to House2Order. We begin the tutorial with the discussion of the meta-models that describe the input and output domains of the transformation. Both meta-models used during the transformation are very simple.

2. Overview of the two meta-models

The house meta-model (located by default in Program Files\ISIS\GReAT\Samples\House2Order\Meta) allows us to build models of houses. The Order meta-model (located by default in the same location) allows us to build purchase-orders for various parts. Let's take a look at both meta-models.

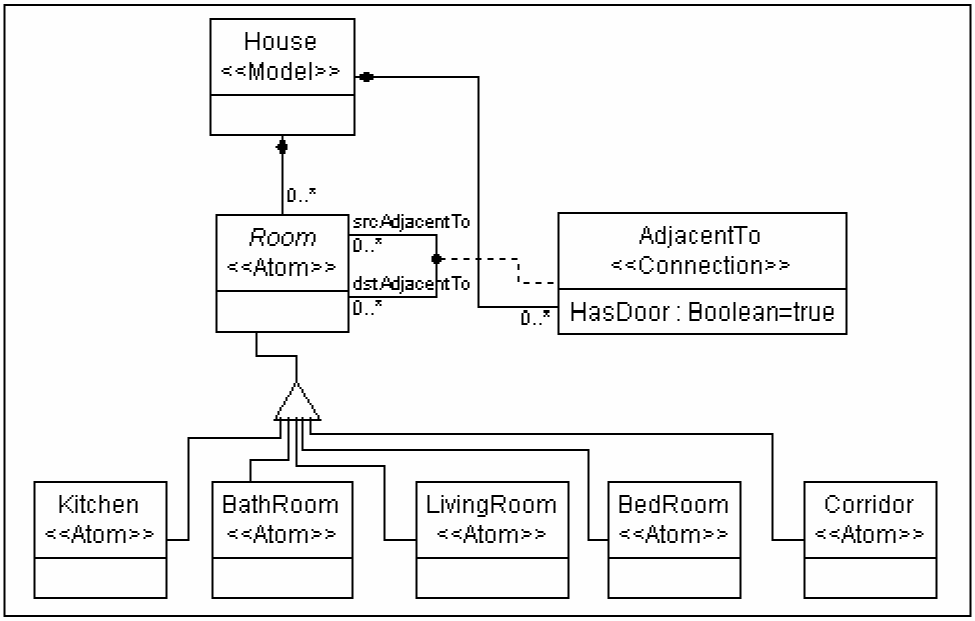


Figure 1 The House Meta-Model

As you can see from the figure above, we can have houses contained in the root folder (House has the attribute "In Root Folder?" set to true, but this is not shown in the above diagram), and we can have as many types of rooms in the house as we want. In addition, we can have a connection between two rooms if they are adjacent, and if the rooms have a door between them, then we set the Boolean attribute "HasDoor" of our connection ("AdjacentTo") to true, otherwise it is false.

The following figure shows the Order meta-model.

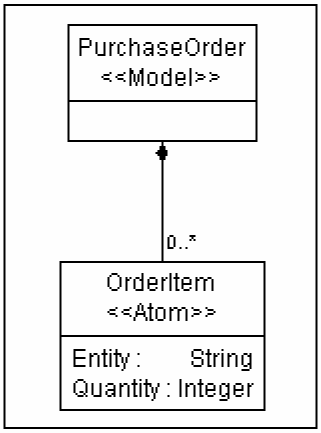


Figure 2 The Order Meta-Model

As you can see from this very simple meta-model, we can have PurchaseOrders in the root folder (again, not shown in this model), and each purchase order can contain any number of OrderItems – each OrderItem has a string identifying the type of item, along with the number of items to order.

3. Purpose of transformation

The purpose of this transformation is to count the number of "AdjacentTo" connections inside an input model conforming to the House meta-model that have the Boolean attribute "HasDoor" set to true. We want to create an output model conforming to the Order meta-model that contains a PurchaseOrder for each House in our input model (remember, any number of Houses can be contained in the root folder of the input model), along with an OrderItem describing how many doors to order if the House does contain "AdjacentTo" connections that have doors.

Now that we know what we want to do, let's go through the process of setting up the GReAT Transformation step-by-step.

4. The House2Order GReAT Transformation

4.1. Step 1 – Create an empty UMT Project

Create a new UMLModelTransformer project in GME.

1. Go to GME, select "New Project…" from the "File" menu.
2. Select UMLModelTransformer as the paradigm. Click "Create New"
3. Select "Create project file" and click "Next"
4. Give the project a name and select, "Open".
5. Save the project without doing anything and close the project.

In this tutorial, I'll assume that in step 4 above, you have named this UMLModelTransformer project "MyHouse2Order.mga".

4.2. Step 2 – Attach GME meta-models

Attach the GME-style meta-models of the input and output models to your UMLModelTransformer project in UML-style.
In GReAT, you specify patterns by using references to UML classes from your meta-models. However, GReAT is built on top of UDM (Universal Data model), and UDM works with meta-models in the UML paradigm, so we need to convert your GME-style meta-model(s) into UDM-style meta-models.

1. Open the HouseModel meta-model in GME (located by default in Program Files\ISIS\GReAT\Samples\House2Order\Meta)
2. Run the interpreter that says, "Converts GME metamodel to UML class diagram in a given UMT paradigm" (this is shown in the figure below).
3. When the "Save As" dialog pops up, select the UMLModelTransformer project you created in Step 1 above (MyHouse2Order.mga). This allows us to attach the UML-style meta-model to the UML Model Transformer project we created earlier.
4. A dialog box will pop up saying that "MyHouse2Order.mga" has been created\modified and that changes have been made to this metamodel. **At this point, you need to run the MetaGME interpreter!** Although you haven't made any changes to the metamodel, there may have been some rolename changes made when the UMLModelTransformer interpreter was run. These won't affect any existing models, but you need to run the MetaGME interpreter in order for your GReAT transformation to succeed later. You can run the MetaGME interpreter by going to "File -> Run Interpreter -> MetaGME2004 Interpreter". Accept the defaults, and register the paradigm when asked.

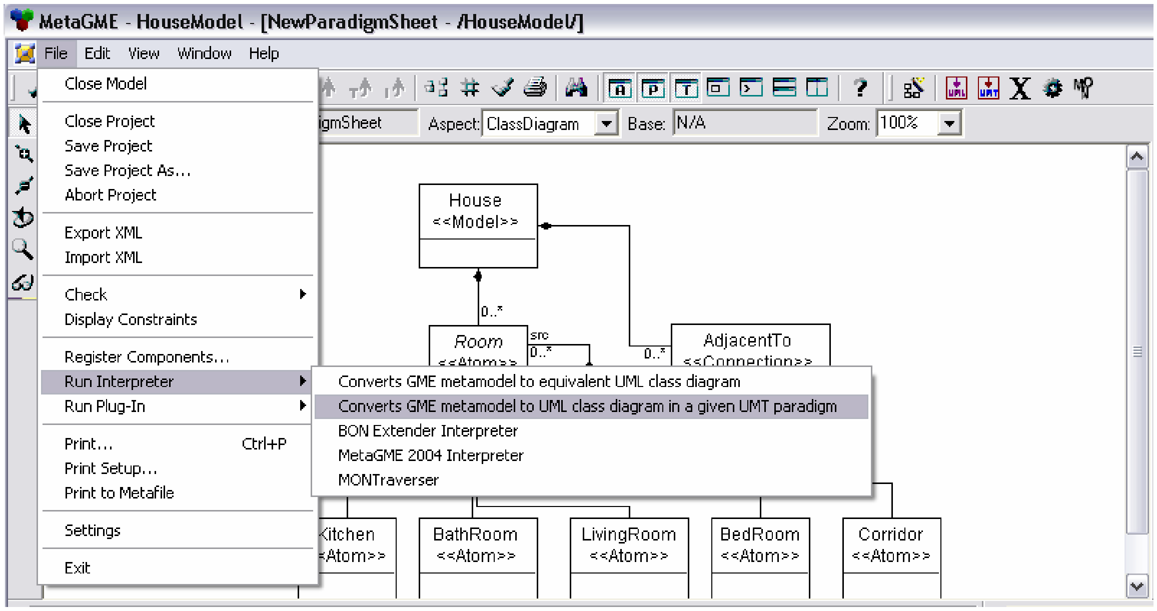


Figure 3 Interpreter to Invoke

5. Close the HouseModel meta-model, and open the Order meta-model (again, located by default in Program Files\ISIS\GReAT\Samples\House2Order\Meta). Follow the same steps with this meta-model as above (run the “Converts GME metamodel to UML class diagram in a given UMT paradigm” interpreter, select “MyHouse2Order.mga”, and run then MetaGME interpreter).

6. Open your transformation project file (MyHouse2Order.mga). In the Browser window, you should see something like the following screenshot:

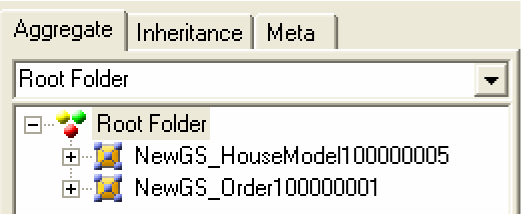


Figure 4 Old Names

7. Remove both the “New” from the beginning of the names of the meta-models, and remove the numbers at the end. These are added so that you can update your meta-models if you change them later. The browser should now look like the following:

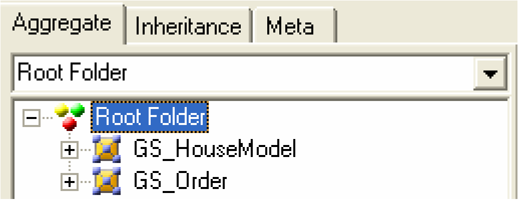


Figure 5 New Names

4.3. Step 3 – Insert a configuration

1. Right click on the "Root Folder" object in the browser and go to Insert Folder >> Configurations. Your browser should now look like the following.

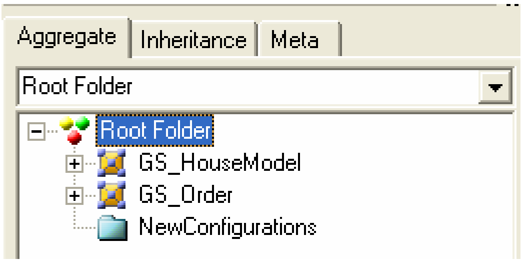


Figure 6 Current Browser

2. Right click on the "NewConfigurations" folder and select Insert Model -> Configuration.
3. Open this newly inserted configuration model. The part browser for objects you can insert should look like the following:

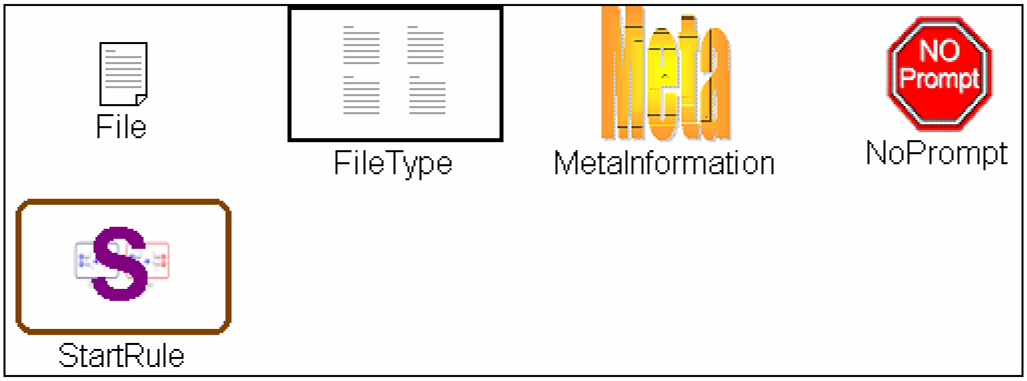


Figure 7 Parts to insert into a configuration

Here's an brief explanation for the 3 things we'll be using at this time:

1. File – This gives the name and path to the input models. We connect this to a FileType object which is described below.

2. FileType – This gives the meta-model of the input file attached, along with various other information
3. MetaInformation – This lists the UDM project file used by the transformation along with the GR file (which contains the re-writing rules in an internal format used by GReAT).

4.4. Step 4 Insert Elements into Configuration

1. Insert a File object and name it "HouseFile". This represents the House model that we are going to be using to generate our Order model. In the Attribute Panel, give the path to the actual .mga file (if it is in the same directory, you can specify something such as, "HouseInput.mga".
2. Insert a FileType object and name it, "HouseFileType". This represents various information about the HouseFile object created above. In the attribute panel, set the following information:
    a. Meta name should be GS_HouseModel (must be the same as the name of the House meta-model in the browser.)
    b. Root class name should be RootFolder (The input model is a GME model, so everything is contained inside the RootFolder).
    c. File mode should be Read since this file already exists and we won't be modifying it (if we were modifying it, we would need to open it "Read and write" or "Read and write to a copy").
    d. DTD pathname should be Udm\ (this attribute is relevant only when using XML files to store the models)
    e. Run transformation in memory can be true or false – this transformation is small and simple, so false is fine (setting this attribute to true will result in copying the model files into memory prior and after the transformation; it is advisory to use this feature when executing large transformations, i.e. when the reduced runtime of the transformation process outweighs the cost of copy operations.).
3. Double click on HouseFileType. Now insert a FileObject, and in the Attribute Panel set the ObjectPath to RootFolder; .There should be a semi-colon at the end of RootFolder. We'll explain more about this ObjectPath object later.
4. Go into "Connection" mode and connect the HouseFile to the HouseFileType.
5. Your configuration should now look like the following:

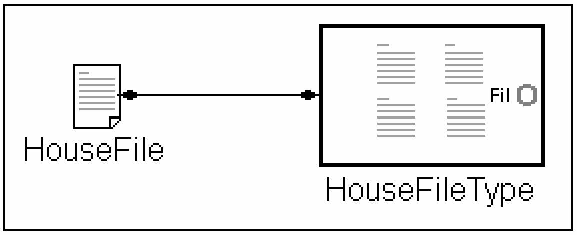
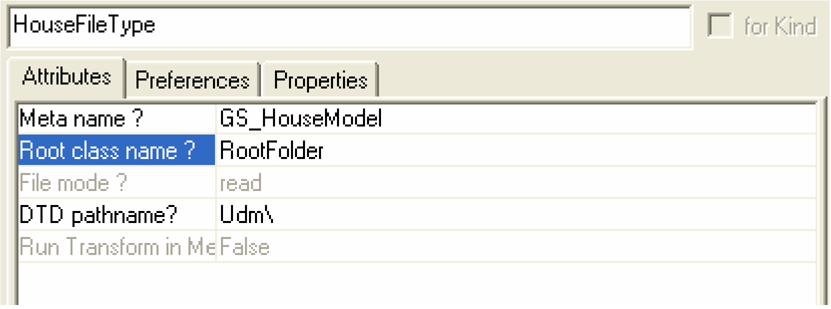


Figure 8 Current Configuration

and the HouseFileType attributes should look like the following:

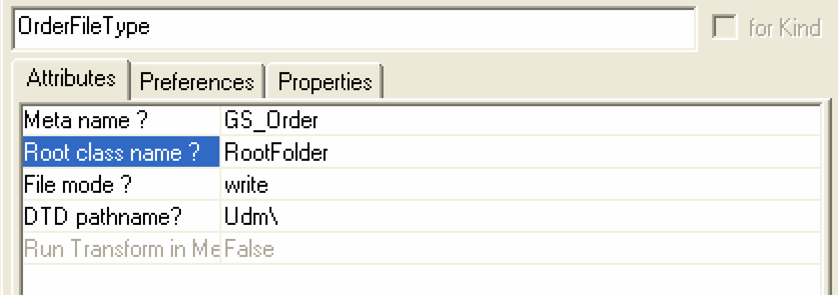


HouseFileType

Meta name ?	GS_HouseModel
Root class name ?	RootFolder
File mode ?	read
DTD pathname?	Udm\
Run Transform in Me	False

Figure 9 HouseFileType's attributes

6. Now we need to add a File object for the Order model we will be creating. Drag a File object onto the configuration and name it "OrderFile". Set its File path name to "Output.mga".
7. Add a FileType object for our OrderFile object created above in step 6. Name it OrderFileType and set its attributes as shown in the next figure.



OrderFileType

Meta name ?	GS_Order
Root class name ?	RootFolder
File mode ?	write
DTD pathname?	Udm\
Run Transform in Me	False

Figure 10 OrderFileType's Attributes

IMPORTANT NOTE: Make sure that the "File mode" attribute is set to "write" for any file that does not already exist; otherwise, you will get an error when you try to run the transformation.

8. Double click on OrderFileType. Now insert a FileObject, and in the Attribute Panel set the ObjectPath to RootFolder; There should be a semi-colon at the end of RootFolder.
9. Connect OrderFile to OrderFileType.
10. Insert a MetaInformation object into the configuration. We don't need to fill its attributes in, they will be completed automatically when we invoke the transformation for the first time.
11. The entire configuration should now look like the following:

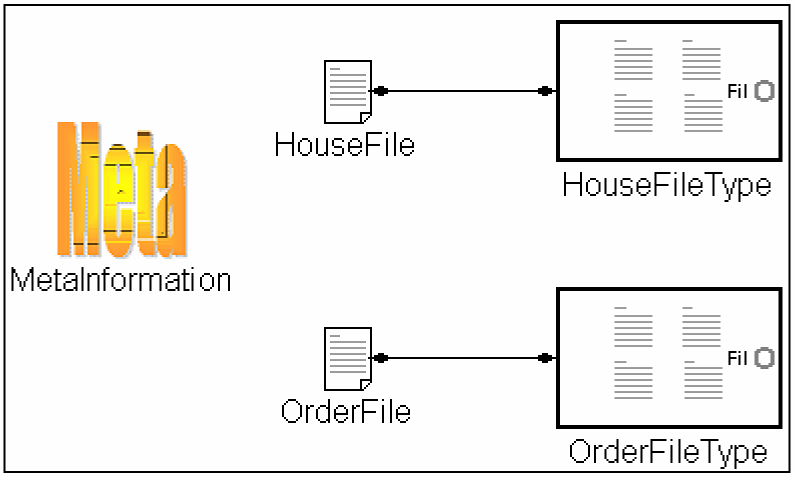
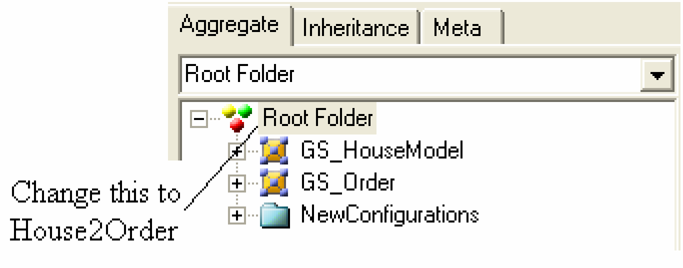


Figure 11 Current view of the configuration

12. One last bit with the configuration and we are done. Change the name of the root folder to something meaningful as shown in the following figure.



Change this to House2Order

Figure 12 Invalid root folder name

Note, that the name of the root folder must not contain spaces. The Browser should now look like the following.

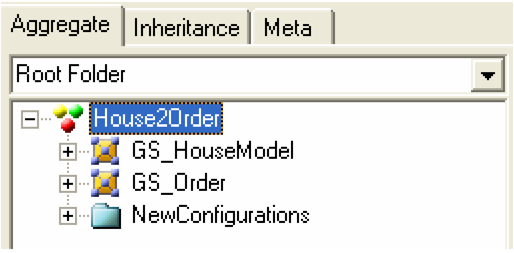


Figure 13 Root folder without spaces

We are now ready to give the rules for the transformation.

4.5. Step 5 – Specify the transformation

1. Right click on the root folder (which is now named “House2Order”), select “Insert Folder” and then select “Transformation” as shown in the figure below.

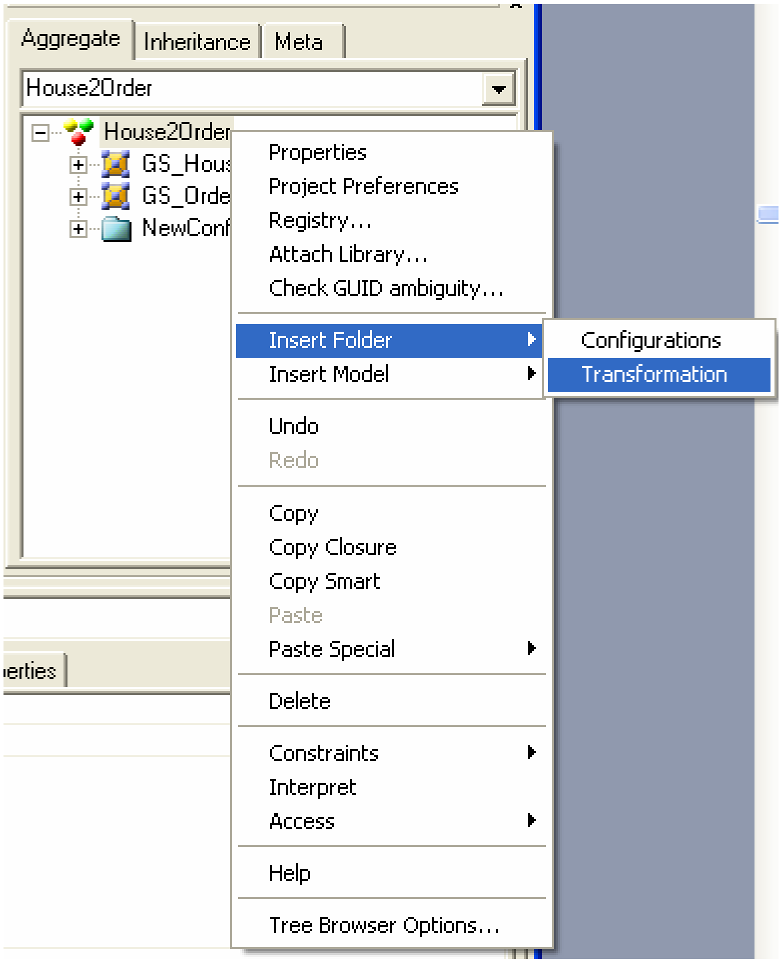


Figure 14 How to insert a Transformation folder

Your browser should now look like the following:

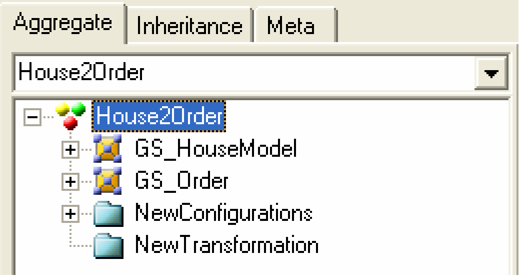


Figure 15 Browser with Transformation Folder

2. Right click on the “NewTransformation” folder, go to “Insert Model” and select “Block” as shown in the next figure.

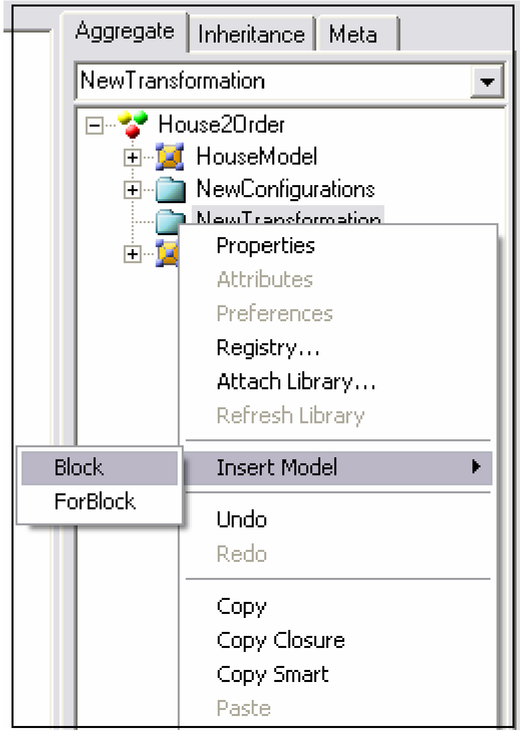


Figure 16 How to insert a block

The browser should now look like the following:

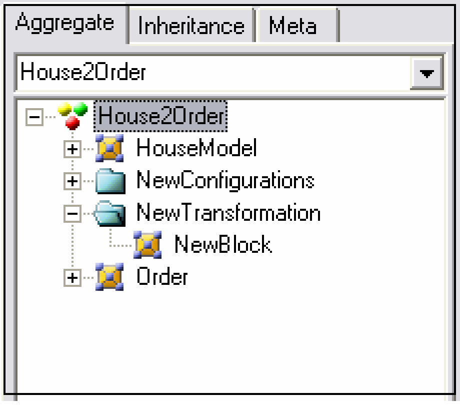


Figure 17 Updated Browser

3. Open the NewBlock by double-clicking it in the browser. You should see a blank sheet appear in the main GME window, and your part browser should look like the following:

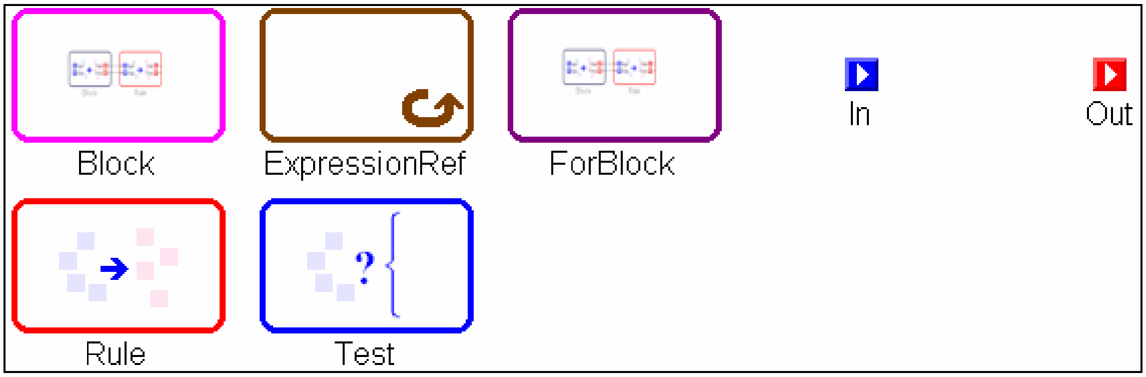


Figure 18 Part Browser for a Block

The following briefly explains the parts we'll be using at this time:

  1. Rule – The basic rewriting unit. Each rule contains a graph pattern composed from UML classes.
  2. ExpressionRef – These are references to rules that have already been defined; by using these, you can express recursion in your transformation.
  3. Block and ForBlock – These allow you to group rules together. The difference between the two is in the order incoming packets are passed to the rules which are contained inside. A ForBlock pushes the first incoming packet through all of its contained rules, then pushes the second packet all the way through, etc. A Block pushes all the packets to the first contained rule. The first rule generates a number of packets, which are passed along together to the subsequent rules, etc.
  4. In and Out – These are ports that allow you to pass objects from one rule to another.
  5. Test – Similar to an If/Else construct in textual programming languages.

4. Right now our transformation folder (which is currently named "NewTransformation") contains only a block. We need to insert a Rule if we are going to do any re-writing. We will also need some Input Ports in the block so that we can pass some objects to our first rule. Open the block (named "NewBlock") if it is not already opened by double clicking on it, and drag two Input Ports and a Rule from the Parts Browser into it. NewBlock should now look like the following:

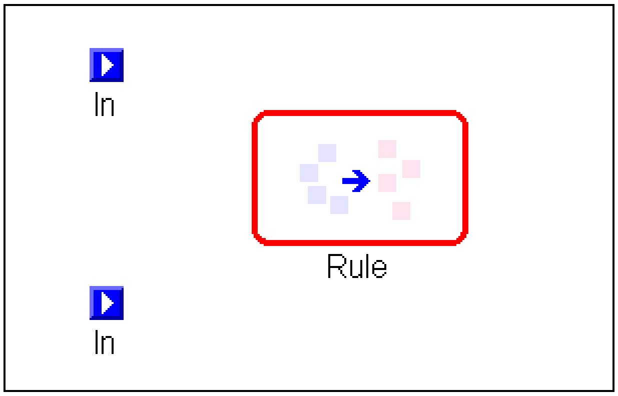


Figure 19 NewBlock after inserting 2 input ports and a Rule

5. Let’s rename the Input Ports and the Rule. Name the top Input Port “HouseIn” and name the bottom Input Port “OrderIn”. Name the Rule “FirstRule”. Your transformation should now look like the following:

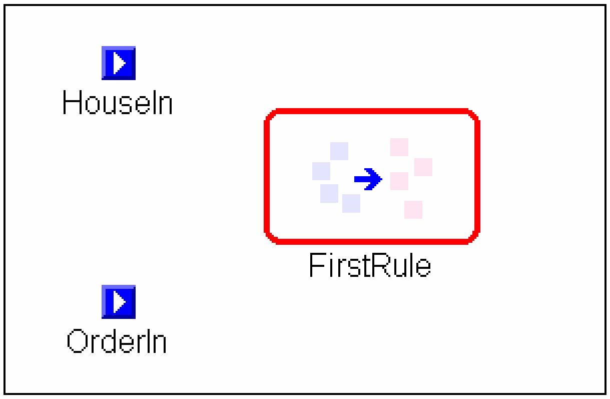


Figure 20 The transformation after renaming elements

6. The two input ports of the block provide us a way to pass objects from our input models to rules contained inside the block. But what do we mean by “objects”? If you remember in our configuration, inside the two filetype objects (named “HouseFileType” and “OrderFileType”) we inserted a FileObject and set the ObjectPath attribute to “RootFolder;”. FileObjects let us select model objects in a containment hierarchy by specifying their types (starting from the root container), and optional “attribute name=value” expressions. In our example, they allow us to pass the RootFolder of our input files to the first Block of our transformation.
7. Open the configuration model (by double-clicking on “NewConfiguration” in the browser window. It should look like the following (the same as earlier):

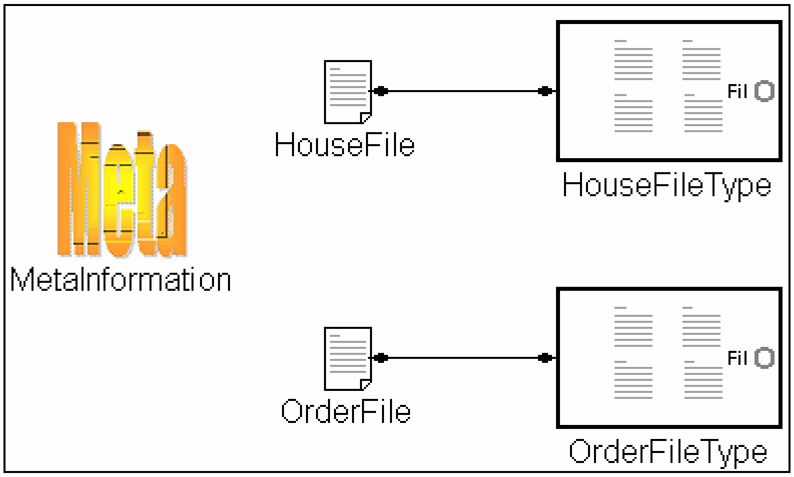


Figure 21 Current view of configuration

8. Now we are going to drag a reference to "NewBlock" onto the configuration. Find "NewBlock" in the browser window, and while holding down ctrl+shift, drag "NewBlock" onto the configuration. You should now see the following:

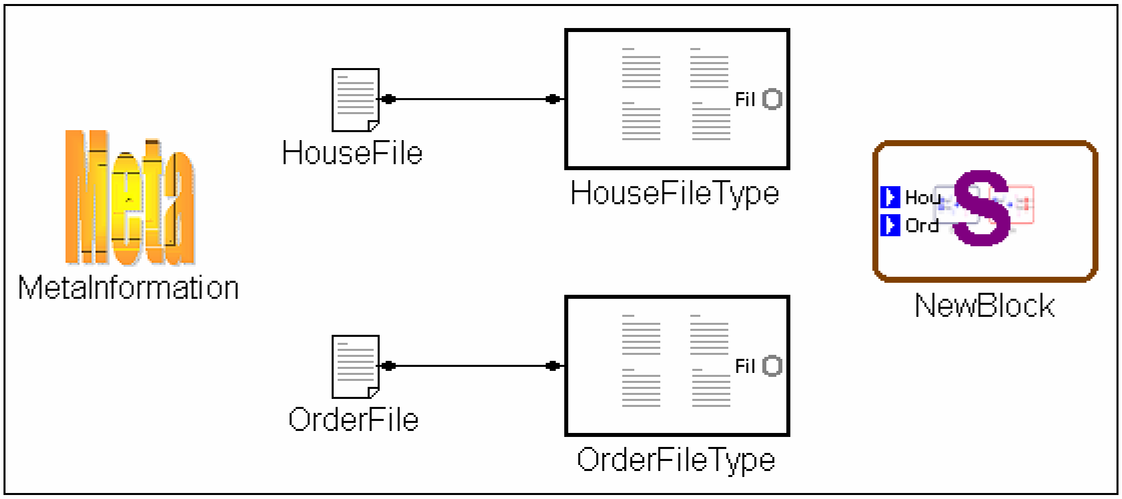


Figure 22 Configuration after inserting a reference to NewBlock

9. Now we are ready to connect the FileObjects to the block. Switch to connection mode inside GME, and connect the FileObject inside "HouseFileType" to the top input port of NewBlock, and connect the FileObject inside "OrderFileType" to the bottom input port of NewBlock. Your configuration should now look like the following:

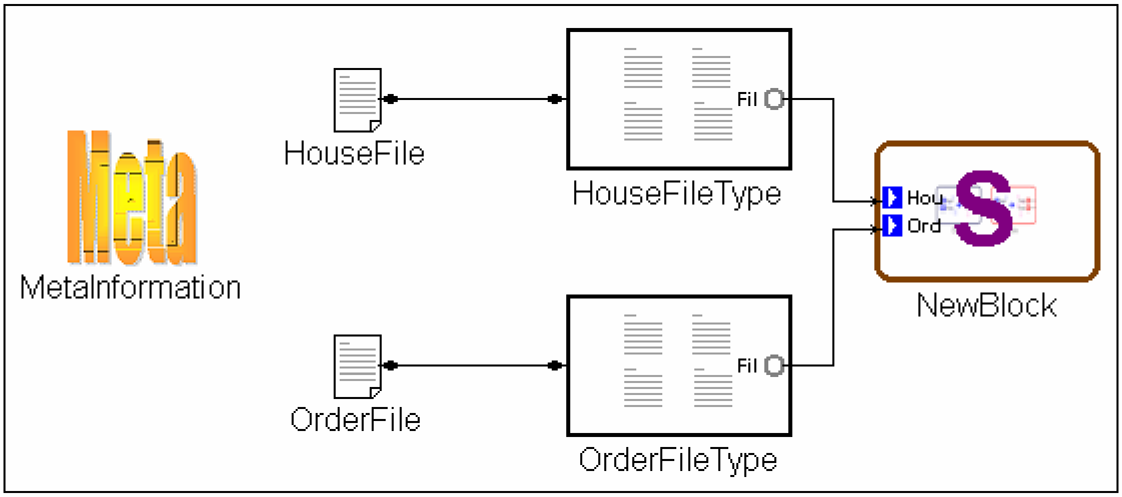


Figure 23 Configuration after connecting the FileObjects to NewBlock

10. Now open up the Rule we created earlier (named "FirstRule"). This rule is blank right now. We need to provide 2 Input Ports in this rule so that we can receive the RootFolder of our HouseModel input model and the RootFolder of our Order input model. Drag 2 Input Ports from the parts browser into the window.
11. We need to tell this rule that what we are passing in are the RootFolders of the HouseModel input model and the Order input model. We do this by dragging a reference to the RootFolders of each into the rule. In the browser, click the + beside the HouseModel, then click the + beside the zC_RootFolderCompositionSheet. Your browser should now look something like the following:

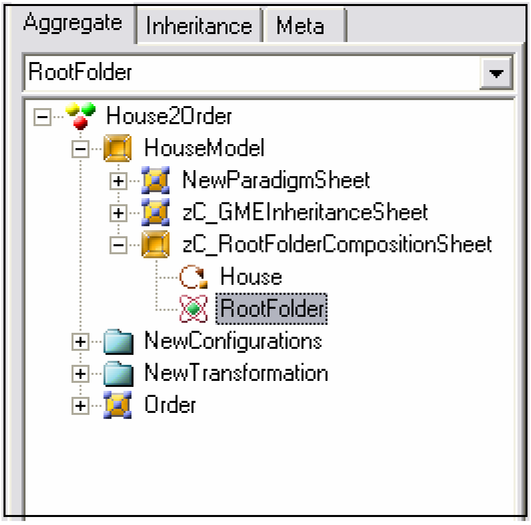


Figure 24 Current view of the browser window

Now, hold down ctrl+shift and drag the RootFolder object (as highlighted in the above diagram) into the rule. Your rule should now look like the following:

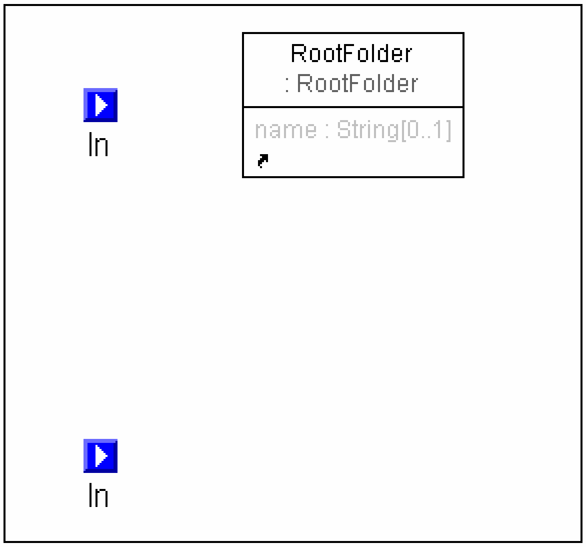


Figure 25 FirstRule after inserting a reference to HouseModel's rootfolder

12. We now need to tell the rule that the top input port will be used for receiving the HouseModel's root folder. Switch to connection mode in GME and connect the top input port to the RootFolder reference. When you do this, a dialog box will appear asking you to select the connection role type. Choose Binding. The rule should now look like the following:

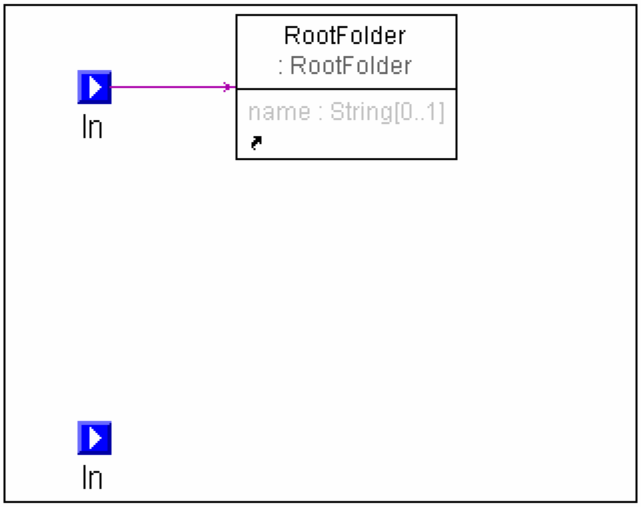
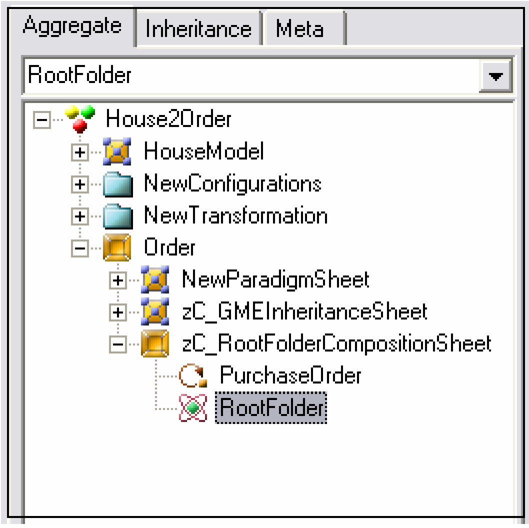


Figure 26 FirstRule after connecting the top input port

13. We now need to insert a reference to the Order metamodel's RootFolder in our Rule. Left click on the + beside "Order" in the browser, then right click beside the + next to zC_RootFolderCompositionSheet. Your browser should look like the following:



Aggregate | Inheritance | Meta

RootFolder

- House2Order
  - HouseModel
  - NewConfigurations
  - NewTransformation
  - Order
    - NewParadigmSheet
    - zC_GMEInheritanceSheet
    - zC_RootFolderCompositionSheet
      - PurchaseOrder
      - RootFolder

Figure 27 Current view of browser window

14. Drag a reference to this RootFolder (as highlighted in the above figure) into the rule by using the ctrl+shift method described earlier. Once you have inserted this reference to the Order meta-model's RootFolder, connect the bottom input port to it (select binding again when the dialog box gives you a choice for connection role type. FirstRule should now look like the following:

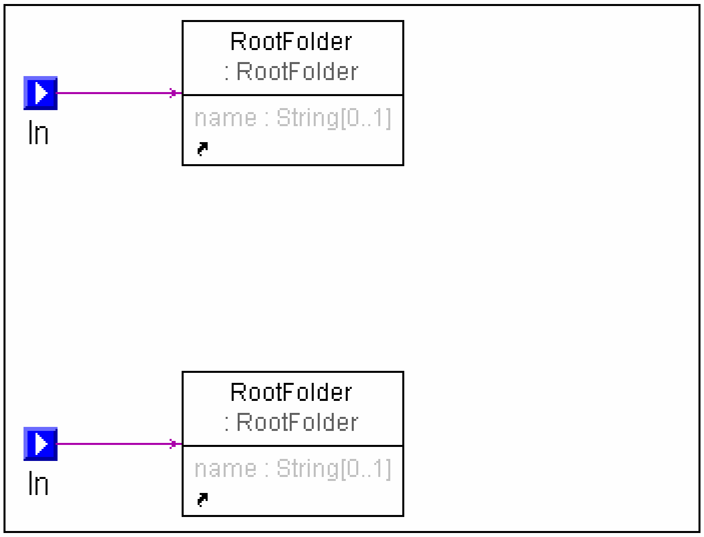


Figure 28 Current view of FirstRule

15. Open "NewBlock". It should look like this:

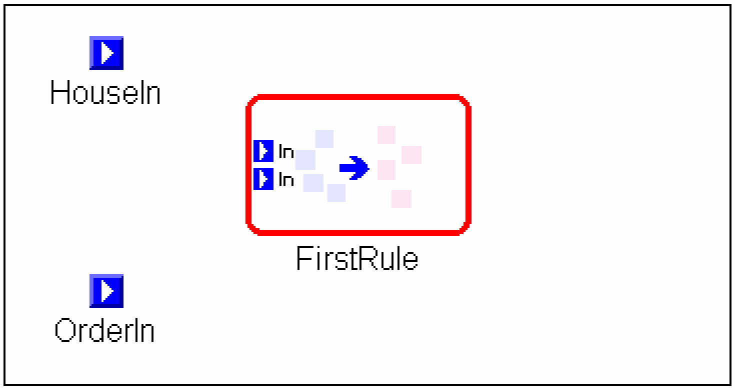


Figure 29 Current view of NewBlock

Switch to connection mode in GME and connect the Input Port “HouseIn” to the top input port of “FirstRule”, and connect the Input Port “OrderIn” to the bottom input port of “FirstRule”. NewBlock should now look like this:

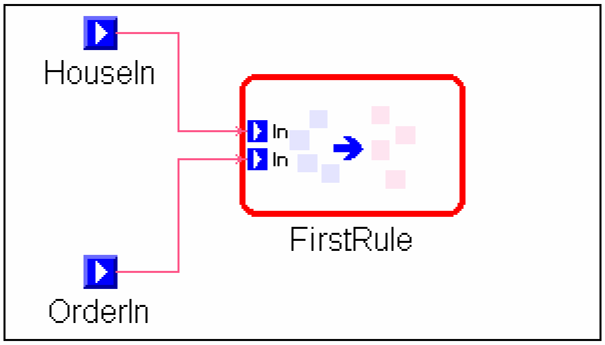


Figure 30 NewBlock after connecting the Input Ports

16. Open FirstRule (which should look like this):

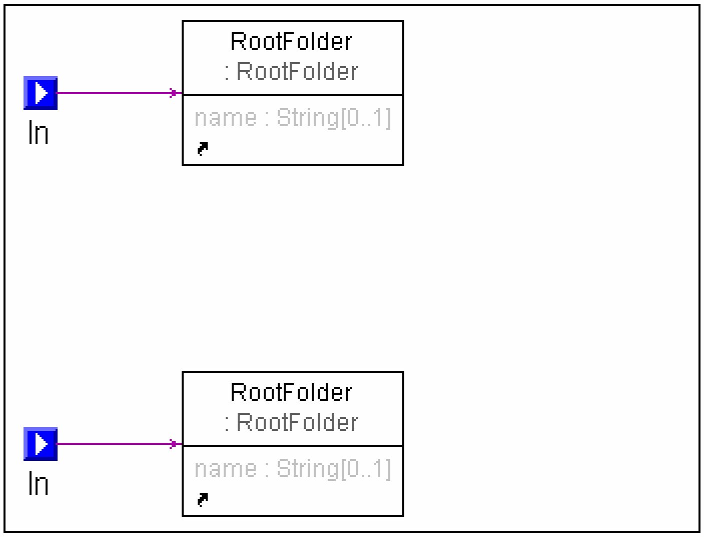


Figure 31 Current view of FirstRule

Let’s examine the current state of our transformation. Right now, upon invocation, the GR-Engine will open the two files we specified earlier in our configuration (HouseInput.mga and Output.mga), and will pass the RootFolder of each of these to FirstRule. Because every valid GME project has only one RootFolder, FirstRule will have only one set of inputs (called a Packet), and thus will execute only once.

Let’s add a pattern for the GR-Engine to match. We stated earlier that we want to create a PurchaseOrder in our output model for every House that is contained in our input model. Let’s specify this pattern that will create a new PurchaseOrder in the RootFolder of Output.mga for every House that is contained in the RootFolder of HouseInput.mga.

17. Expand “HouseModel” in the browser by clicking the + next to it, and then expand “NewParadigmSheet” just below that. The browser should look like this:

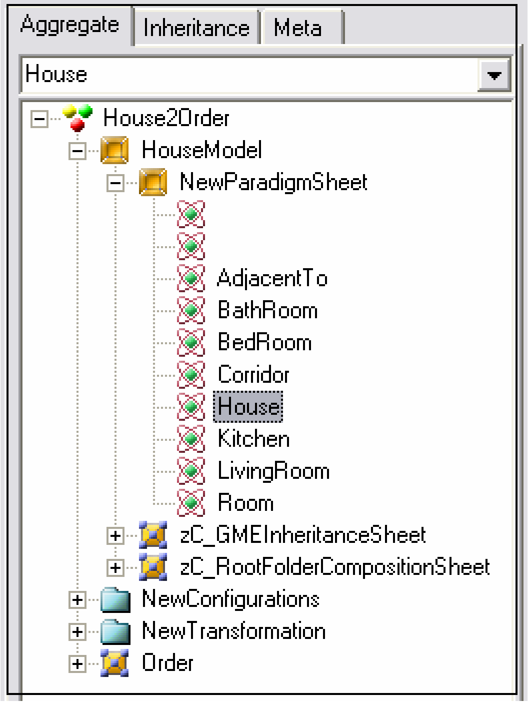


Figure 32 Current view of the browser

18. Drag a reference (using the ctrl+shift method) to House (highlighted in the above diagram) into FirstRule. Now switch to connection mode in GME and connect the reference to House to the top RootFolder (select PatternComposition for the Connection Role Type). FirstRule should now look like this:

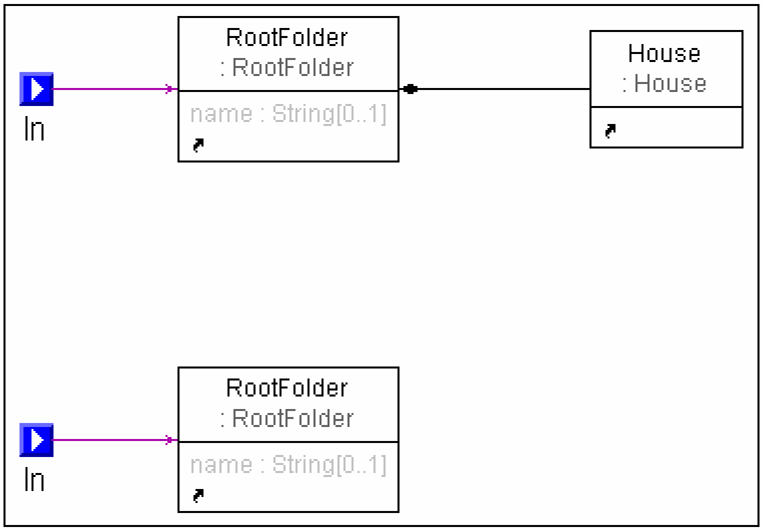
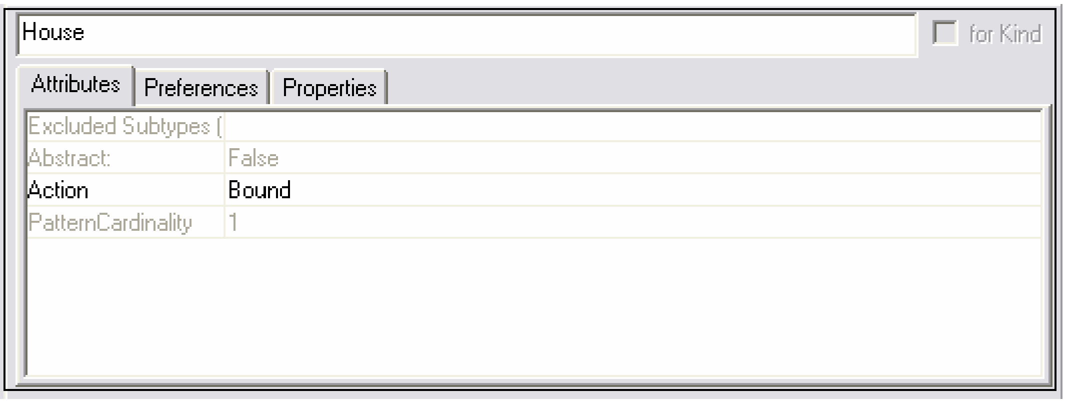


Figure 33 FirstRule after inserting a reference to House

If we click on House and then look at the Attribute Panel, we see the following:



House ☐ for Kind

Excluded Subtypes (	
Abstract:	False
Action	Bound
PatternCardinality	1

Figure 34 The attribute panel with House selected

The important thing to note here is that the attribute "Action" is set to "Bound". This means that we are looking for Houses in the RootFolder of HouseInput.mga (that is, we expect them to already exist). Other options for "Action" are "CreateNew" and "Delete". When a rule is executed, we first look for all matches where the objects are marked as "Bound". If we find any matches, then any objects marked as "Delete" are removed, and any objects marked as "CreateNew" are created. In the next step we will mark an object as "CreateNew".

19. Click on the + beside Order in the browser window, and then click on the + next to NewParadigmSheet just below it. The browser should now look like the following:

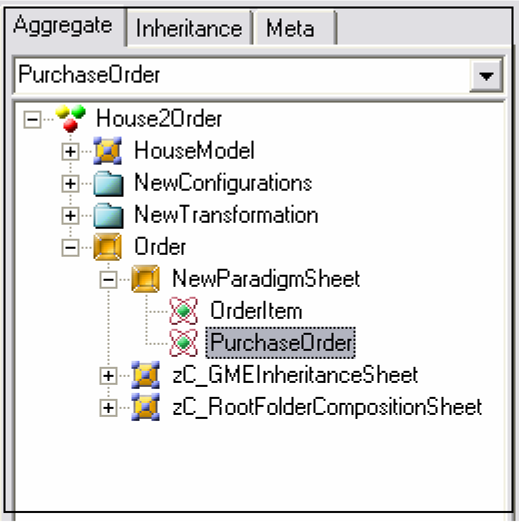


Figure 35 Current view of Browser Window

Drag a reference to PurchaseOrder (as highlighted in the figure above) into FirstRule, and connect this PurchaseOrder to the bottom RootFolder (select PatternComposition for the Connection Role Type). Also drag two Output Ports from the Parts Browser into FirstRule. FirstRule should now look like this:

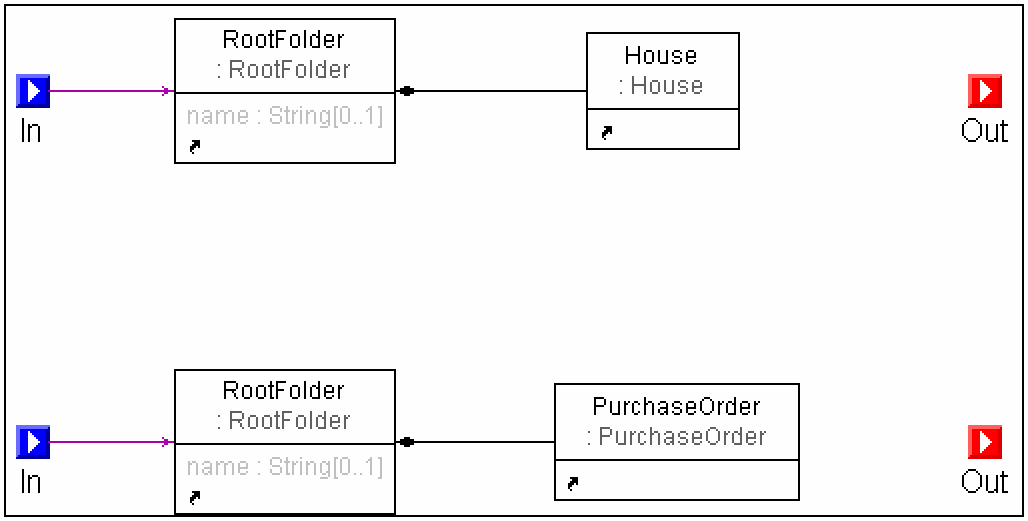


Figure 36 Current view of FirstRule

20. Remember that we wanted to create a PurchaseOrder for every House we found in our input model (HouseInput.mga). Thus, we need to change the "Action" attribute of PurchaseOrder to "CreateNew". Do this by clicking on PurchaseOrder, then selecting "CreateNew" in the Attribute Panel. You will notice that PurchaseOrder (along with its composition link) turns blue to indicate that it is being created. FirstRule now looks like this:

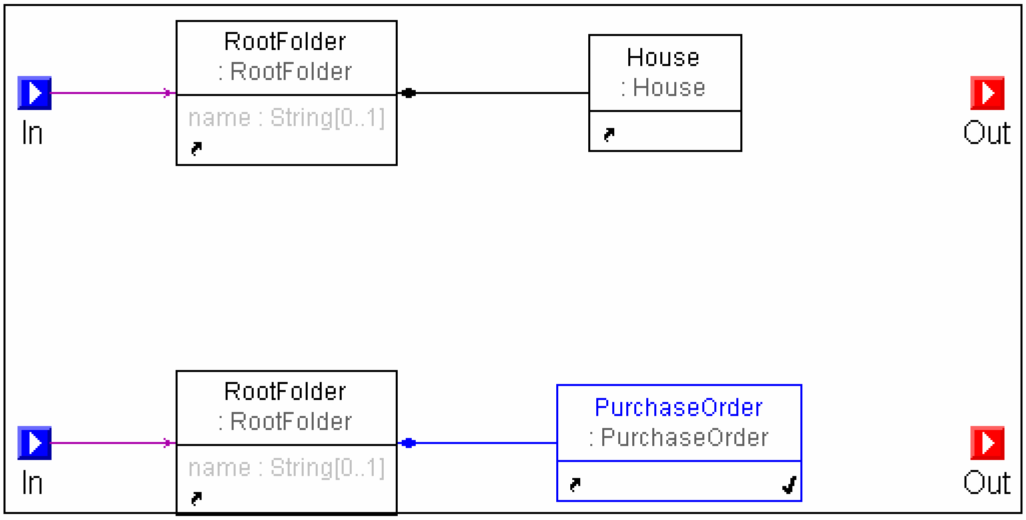


Figure 37 Current view of FirstRule

IMPORTANT NOTE: If the containment link connecting PurchaseOrder to RootFolder does not automatically turn blue, click on the containment link and make sure that in the attribute panel its "Action" attribute is set to "CreateNew".

21. This is all we wanted to accomplish in this Rule; we'll do the other re-writing in other rules. However, let's pass the House objects and the newly created PurchaseOrder objects to the next rule. Switch to connection mode in GME and connect House to the top output port, and connect PurchaseOrder to the bottom output port. FirstRule now looks like this:

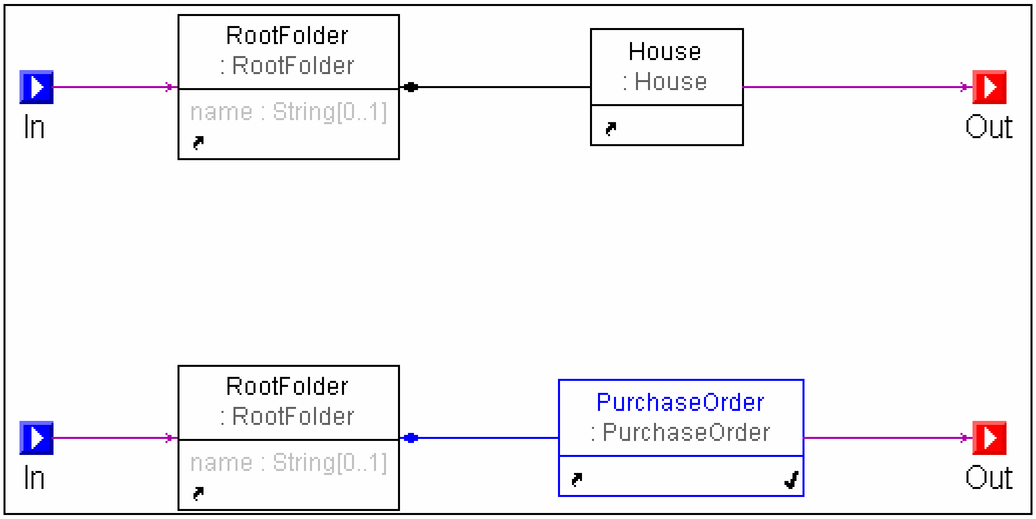


Figure 38 Completed FirstRule

22. Open NewBlock and drag in a new rule from the Parts Browser. Name this Rule "SecondRule". Insert two Input Ports into SecondRule, and then connect the

Output Ports of “FirstRule” to the Input Ports of “SecondRule”. “NewBlock” should now look like the following:

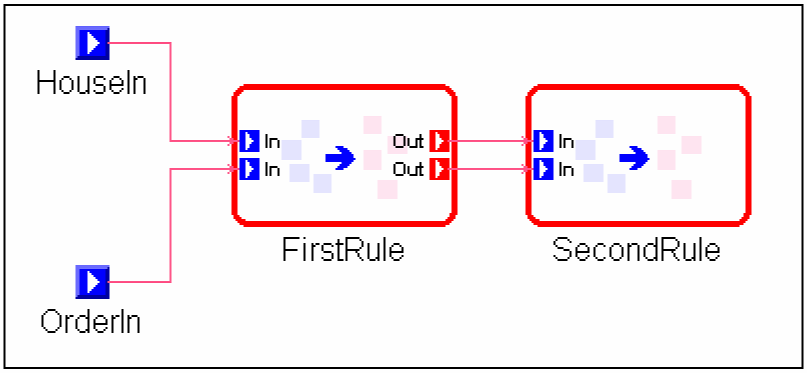


Figure 39 NewBlock after connecting FirstRule and SecondRule

23. Open SecondRule. It currently only contains two Input Ports. Because SecondRule is getting its input from FirstRule, the InputPorts of SecondRule must be bound to the same type of objects as are being passed from FirstRule. From the HouseModel’s NewParadigmSheet, drag a reference to a House into SecondRule, and connect the upper Input Port of SecondRule to this House reference (select Binding for Connection Role Type). Similarly, from the Order’s NewParadigmSheet, drag a reference to a PurchaseOrder into SecondRule, and connect the bottom Input Port to this PurchaseOrder (select Binding again). SecondRule should now look like this:

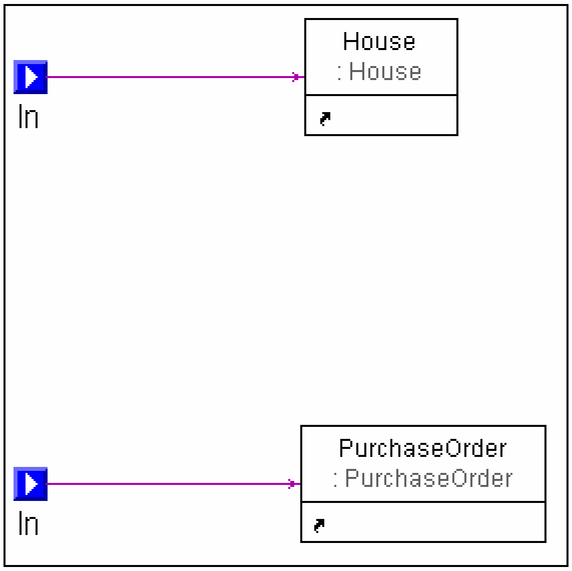


Figure 40 Current view of SecondRule

24. In this rule, we want to create an OrderItem in the PurchaseOrder if and only if there is at least one “AdjacentTo” connection in the corresponding House and that “AdjacentTo” connection has its Boolean attribute “HasDoor” set to true. Thus,

the pattern we are searching for in HouseInput.mga is 2 rooms contained in a House, and an AdjacentTo connection between them. Let's begin specifying this pattern by dragging 2 references of the abstract object "Room" from the HouseModel's NewParadigmSheet into SecondRule, and name one of the rooms "Room1" and the other "Room2". Switch to connection mode, and connect both of these Rooms to House (select PatternComposition for Connection Role Type). SecondRule should now look like this:

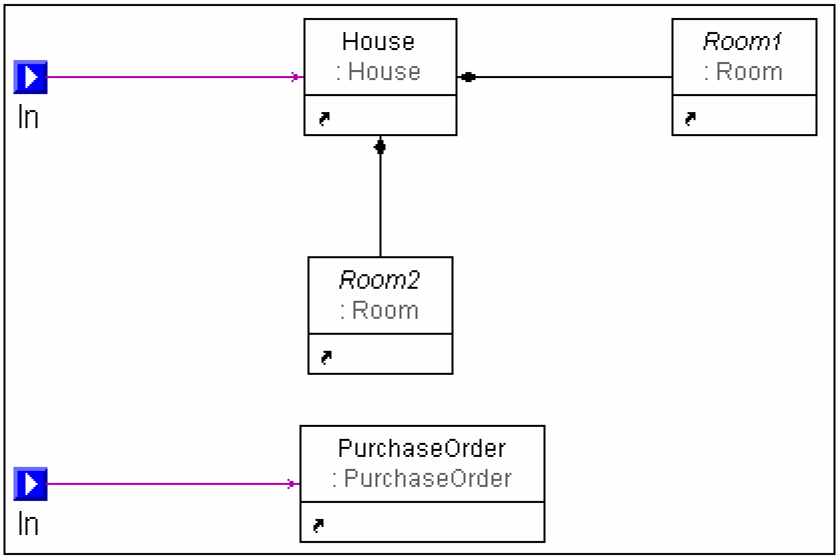


Figure 41 Current view of SecondRule

25. Now, from HouseModel's NewParadigmSheet, drag a reference to "AdjacentTo" into SecondRule.

26. We will now specify that "AdjacentTo" should connect Room1 to Room2. Drag a Connector object from the Parts Browser into SecondRule. Switch to connection mode in GME, and connect "Room1" to the connector object. The rolename next to Room1 is now "src", indicating that Room1 is the first part of the connection. SecondRule should look like this:

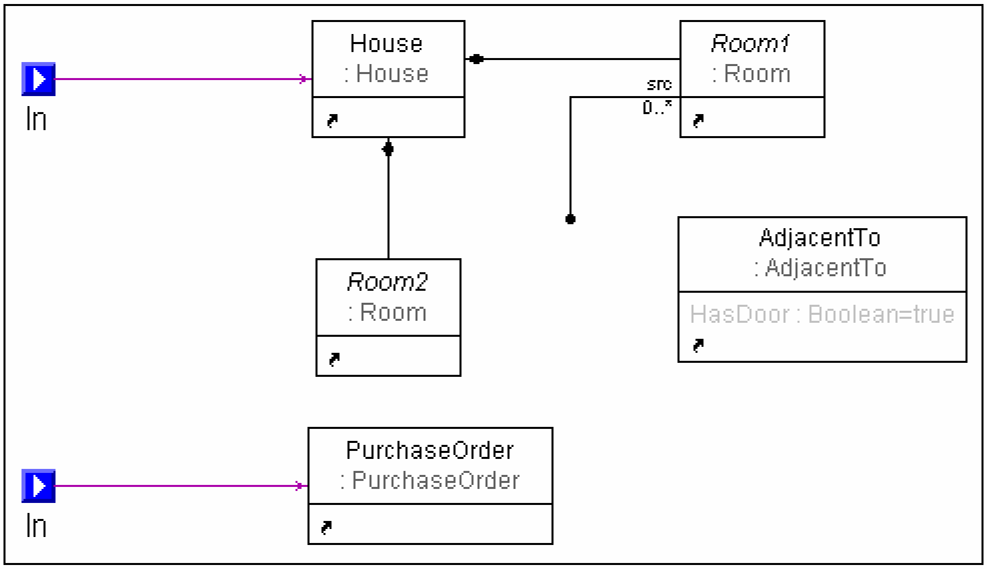


Figure 42 SecondRule after first part of connection specified

27. Now (while still in connection mode) connect the Connector object to Room2 (select Dst for Connection Role Type because Room2 is the destination of the connection). SecondRule now looks like this:

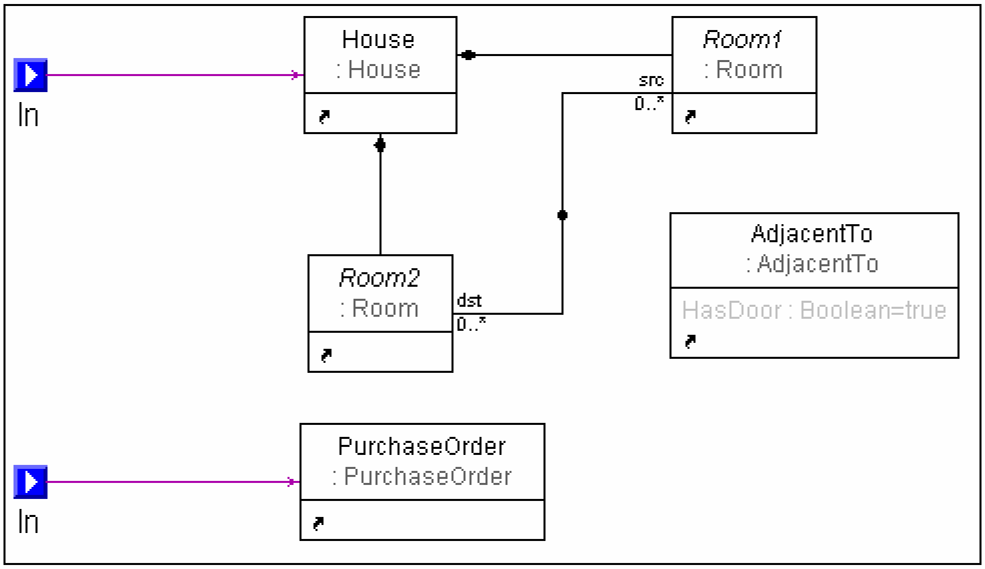


Figure 43 Current view of SecondRule

28. While still in connection mode, connect the Connector object to AdjacentTo (select AssociationClass for Connection Role Type). If you connected the objects to and from the Connector in this order, then you will notice that the source and destination rolenames on Room1 and Room2 change. It is important that these rolenames be the same as theyare in the "NewParadigmSheets". We should also connect AdjacentTo to House (select PatternComposition) to show that the connection AdjacentTo is contained in the House. SecondRule should now look like this:

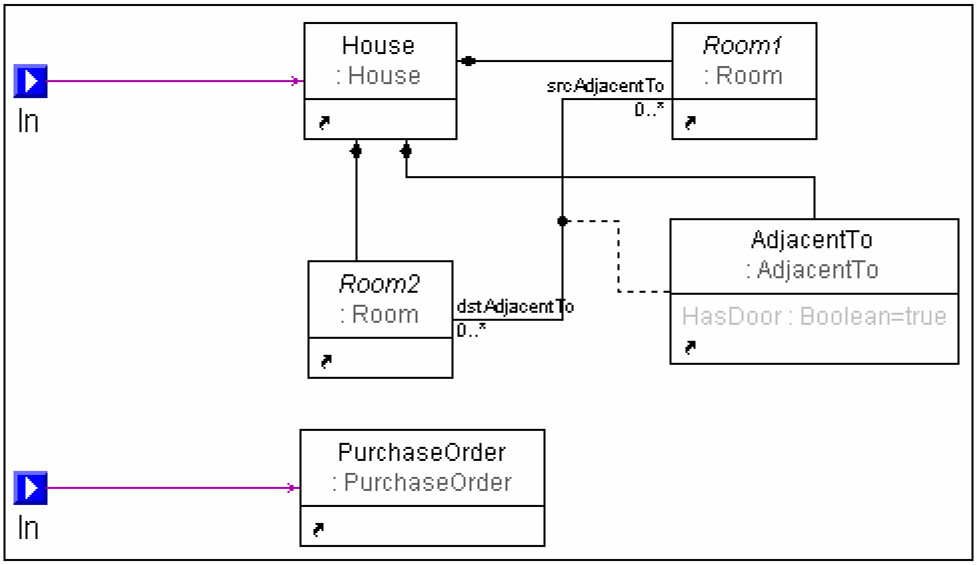


Figure 44 Current view of SecondRule

29. We have specified the pattern we are looking for in HouseInput.mga. Now let's create an OrderItem in PurchaseOrder if we find this pattern. From the NewParadigmSheet under Order in the browser, drag a reference to an OrderItem into SecondRule, and connect it to PurchaseOrder. Also, set the "Action" attribute of this OrderItem to "CreateNew" in the Attribute Panel. OrderItem (along with its pattern composition link) should now both turn blue, and SecondRule should look like this:

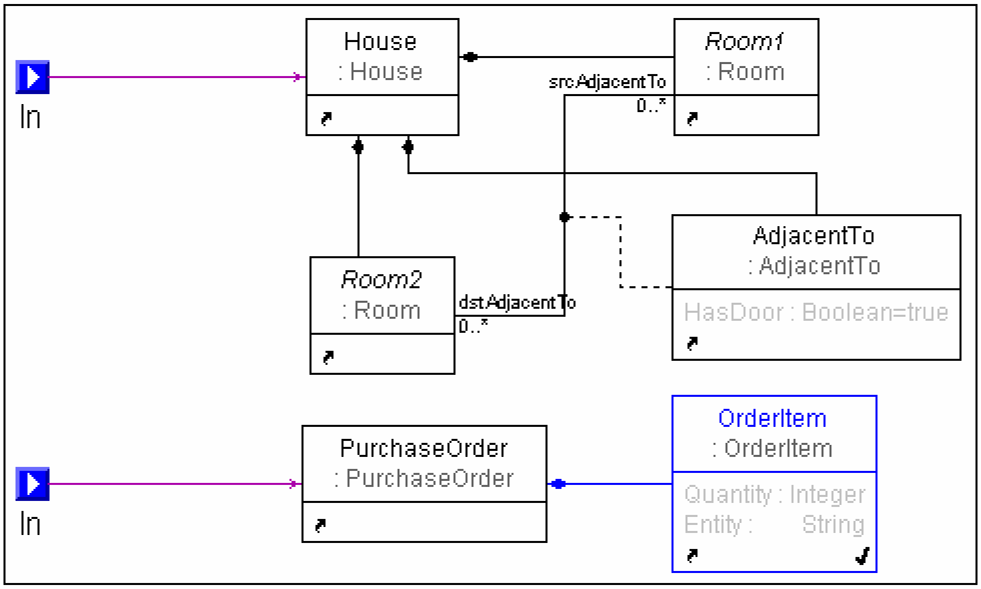


Figure 45 Current view of SecondRule

30. We are almost done with SecondRule. However, we said earlier that we only want to create an OrderItem if the value of AdjacentTo's Boolean attribute

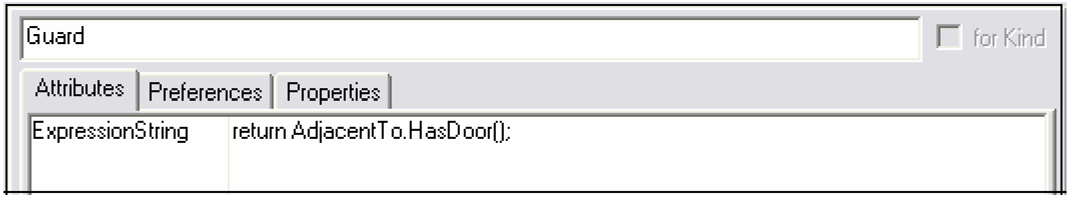
"HasDoor" is true; that is, we only want to create an OrderItem if there is at least one pair of rooms that has a door between them. If the rooms have an AdjacentTo connection connecting them but they don't have a door, then we don't want to find this pattern in the input model. What do we do?

Drag a Guard from the Part Browser into Second Rule (you can drag it anywhere). You can use guards to access the actual attributes of various objects in your model. Guards expressions are specified in C++, and must return with a boolean value. A guard expression evaluates to either true or false; if it evaluates to true, then the match is valid. Otherwise, the match is discarded.

Click on the Guard, and in the Attribute Panel (next to Expression String), insert the following line:

```
return AdjacentTo.HasDoor();
```

This means that we will return true (the match is valid) if the value of HasDoor (the Boolean attribute of AdjacentTo) is true. The Attribute Panel of the Guard is shown below:



ExpressionString return AdjacentTo.HasDoor();

Figure 46 Attribute Panel of Guard

And SecondRule should now look like this:

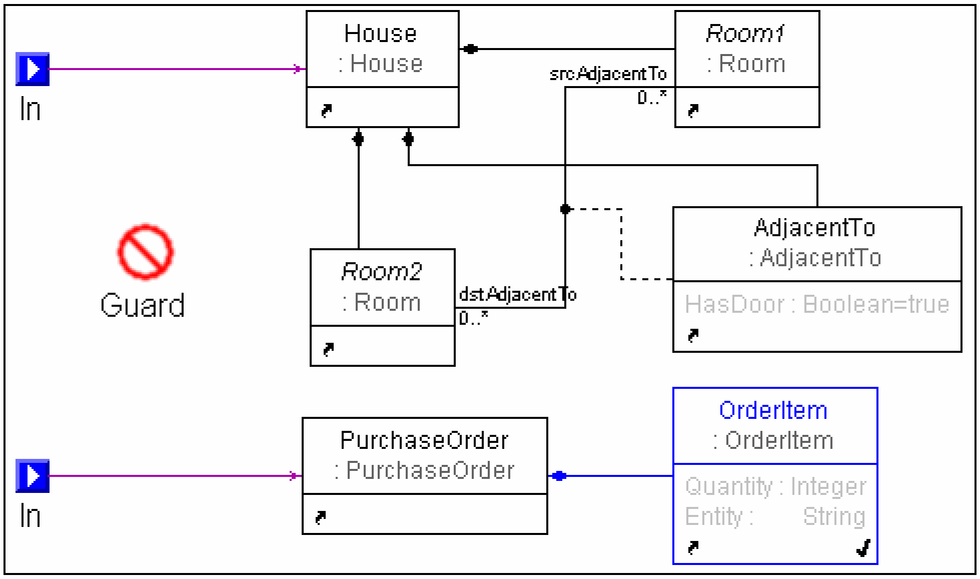


Figure 47 Current view of SecondRule

31. Now we need to pass the House and the newly created OrderItem to the next rule. Drag two Output Ports from the Part Browser into Second Rule, and connect House to the top Output Port, and connect OrderItem to the bottom Output Port. The current view of SecondRule is shown below:

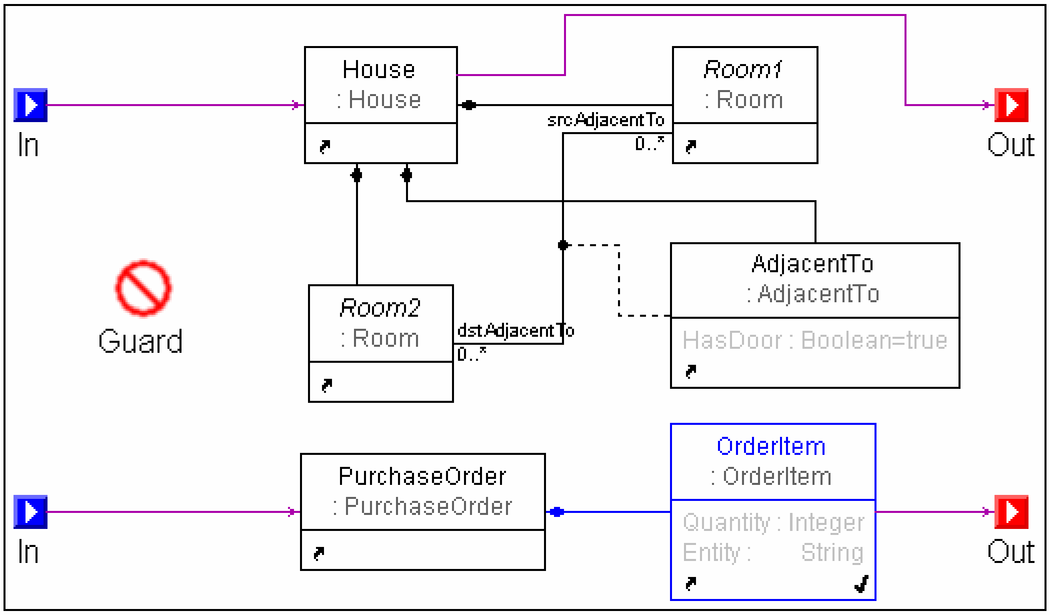


Figure 48 SecondRule with Output Ports

32. We need to initialize the two attributes of the newly created OrderItem. AttributeMapping blocks let you manipulate the attributes of the matched objects by utilizing C++ code snippets. Drag an AttributeMapping block into

SecondRule. Click on the AttributeMapping block, and enter the following into “ExpressionString” in the Attribute Panel:

OrderItem.name() = "Door Order";
OrderItem.Entity() = "Doors";
OrderItem.Quantity() = 0;

As you can probably guess, this sets the initial number of doors to 0, and also allows us to set the name of the OrderItem and its Entity attribute. A screenshot of both the completed SecondRule and the Attribute Panel of the AttributeMapping object are shown below:

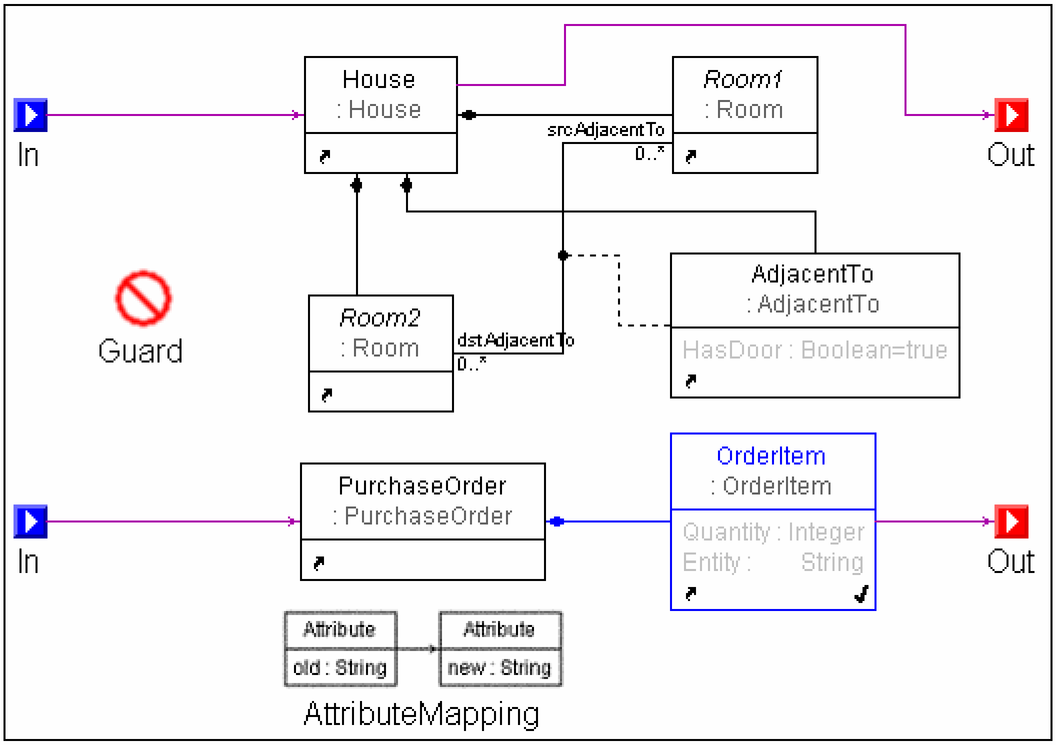
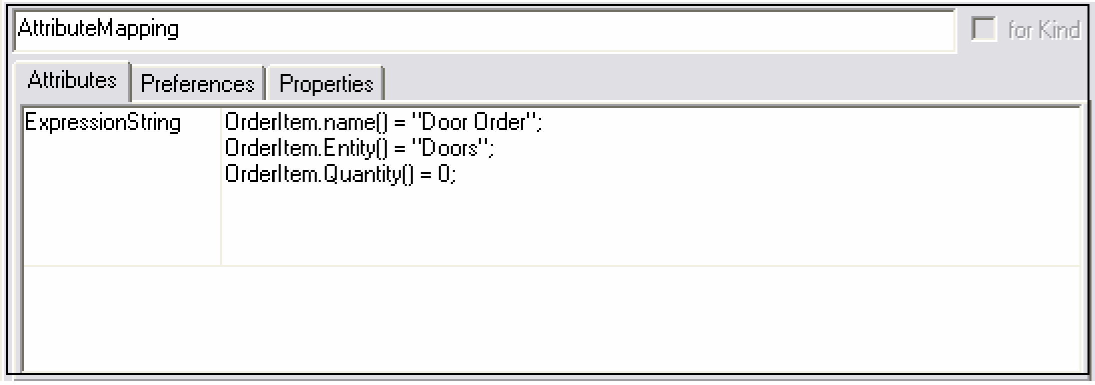


Figure 49 Completed SecondRule



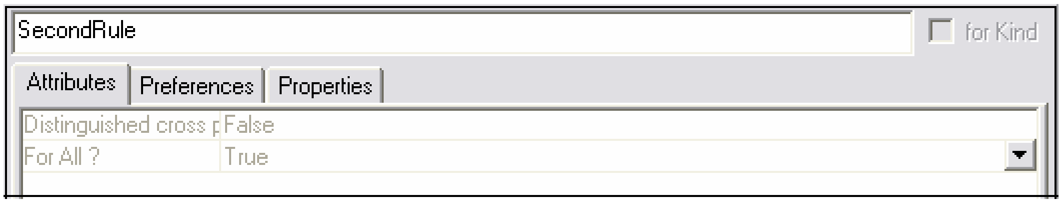
AttributeMapping

ExpressionString OrderItem.name() = "Door Order";
OrderItem.Entity() = "Doors";
OrderItem.Quantity() = 0;

Figure 50 Attribute Panel of the AttributeMapping block

33. There is one slight problem with the rules right now: let's say we have three rooms in a house, roomA, roomB, and roomC, and that there is an AdjacentTo connection with HasDoor set to true between roomA and roomB, and another between roomB and roomC. Currently, SecondRule will find both of these matches, and will create 2 OrderItems in the same PurchaseOrder. However, we only want to create one OrderItem for each House, regardless of how many AdjacentTo connections there are with HasDoor set to true (we will fill in the OrderItem in the next rule).

    There is a solution to this problem. Open "NewBlock" and click on SecondRule. If you look at its Attributes Panel, it should look like the following:

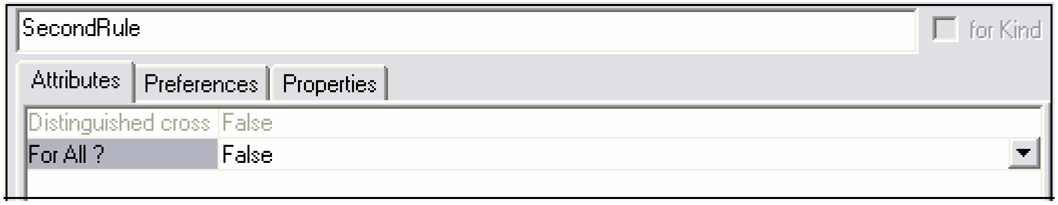


SecondRule

Distinguished cross p False
For All ? True

Figure 51 Attributes Panel for SecondRule

    The attribute ForAll should be changed to false, indicating that for each input packet to the rule, after the first valid match is made, the rule should not fire anymore. After changing the ForAll attribute to false, the Attributes Panel for SecondRule should look like this:



SecondRule

Distinguished cross False
For All ? False

Figure 52 Attributes Panel for SecondRule with ForAll set to False

34. We are now ready to insert the final rule. Open "NewBlock" and drag a new Rule in from the Part Browser, and name the rule "ThirdRule". Open ThirdRule and insert two Input Ports. Open "NewBlock" again and connect the output ports of SecondRule to the Input Ports of ThirdRule. NewBlock should now look like the following:

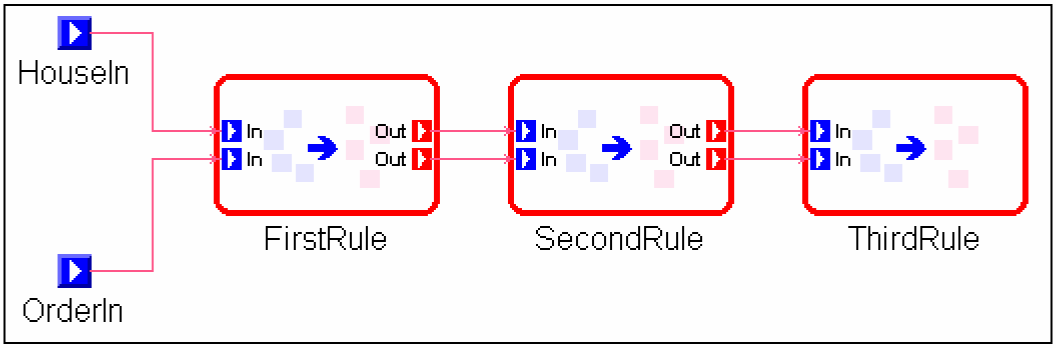


Figure 53 Current view of NewBlock

35. Open ThirdRule. The pattern we are going to specify is almost identical to the pattern we specified in SecondRule. In our input model (HouseInput,mga), we are looking for all "AdjacentTo" connections that have "HasDoor" set to true. We want to find all of these in each house, and we want to increment the number of doors to order in our OrderItem (we want to increment the attribute Quantity by 1). First drag a reference to a House from the House meta-model into ThirdRule, and bind it to the top input port. To create the necessary pattern in the House input model, follow steps 24-28 above (drag in two references to Room objects, rename one to *Room1* and other to *Room2* and connect them both to House, drag in a reference to an AdjacentTo connection and connect it to House, drag in a Connector object (the small black dot) from the Part Browser, and connect Room1 to the Connector, connect the Connector object to Room 2 (choose Dst) and finally connect the Connector object to AdjacentTo (and select Association Class). ThirdRule should now look like this:

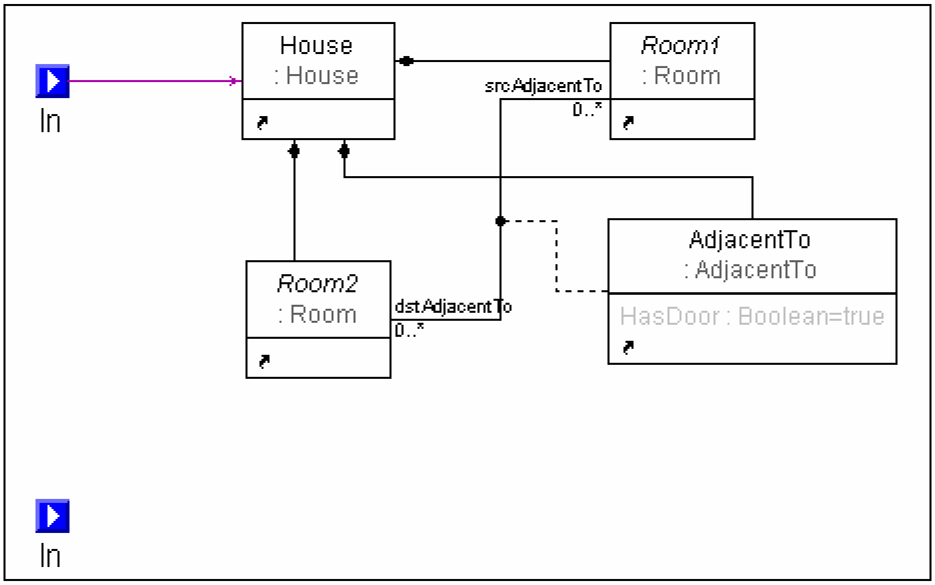


Figure 54 Current view of ThirdRule

36. We passed OrderItems out of the bottom Output Port of Second Rule, so let's drag a reference to an OrderItem (from Order's NewParadigmSheet) into

ThirdRule, and bind this OrderItem to the bottom Input Port. ThirdRule now looks like this:

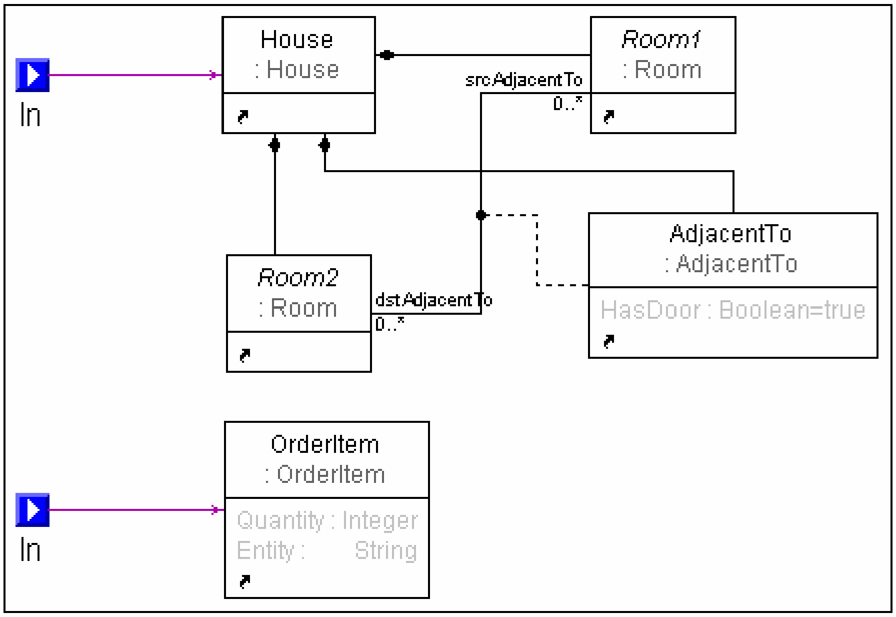
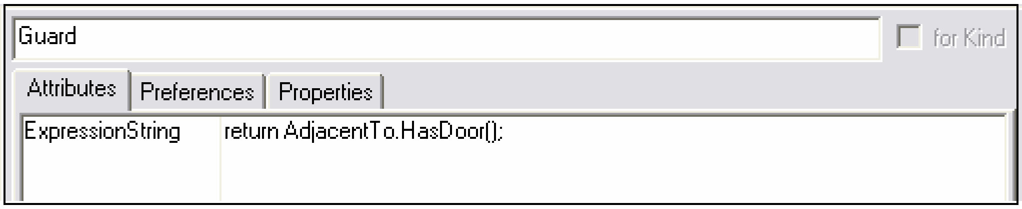


Figure 55 Current view of ThirdRule

37. Only two more things to add and our transformation is complete! Drag in a Guard object from the Part Browser. Click on this Guard, and type the following code into the ExpressionString in the Attribute Browser:

    return AdjacentTo.HasDoor();

    (Remember, code inserted into a Guard or AttributeMapping is case-sensitive) This Guard will allow us to find matches only if the connection between the two rooms has a door. The Attribute Panel of our Guard is shown below:



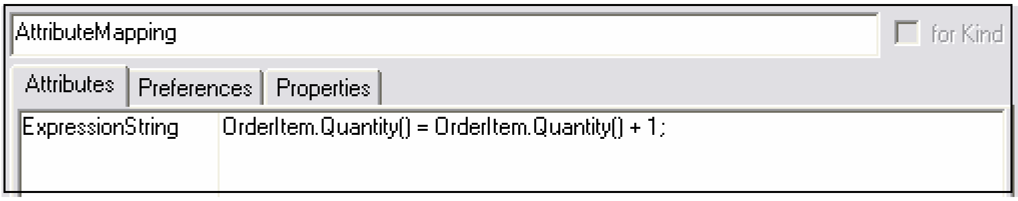
ExpressionString return AdjacentTo.HasDoor();

Figure 56 Attribute Panel of Guard

38. Now drag an AttributeMapping block into ThirdRule. Click on it, and insert the following code into the ExpressionString in the Attribute Browser:

    OrderItem.Quantity() = OrderItem.Quantity() + 1;

This code will increase the Quantity of our OrderItem by 1 every time a match is found. The Attribute Panel of this AttributeMapping is shown below, along with the now completed ThirdRule.



ExpressionString OrderItem.Quantity() = OrderItem.Quantity() + 1;

Figure 57 Attribute Panel of our AttributeMapping

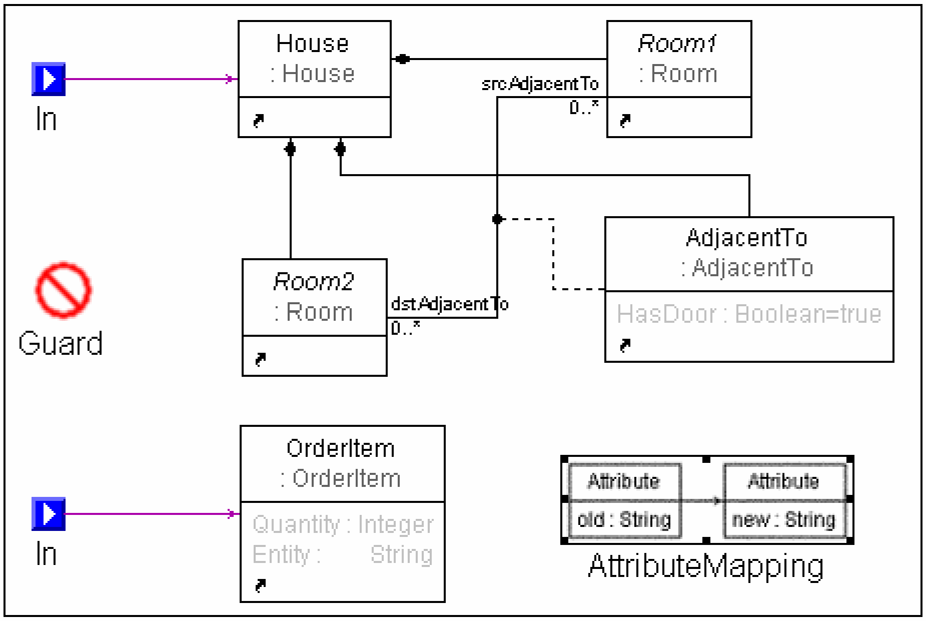


Figure 58 Completed ThirdRule

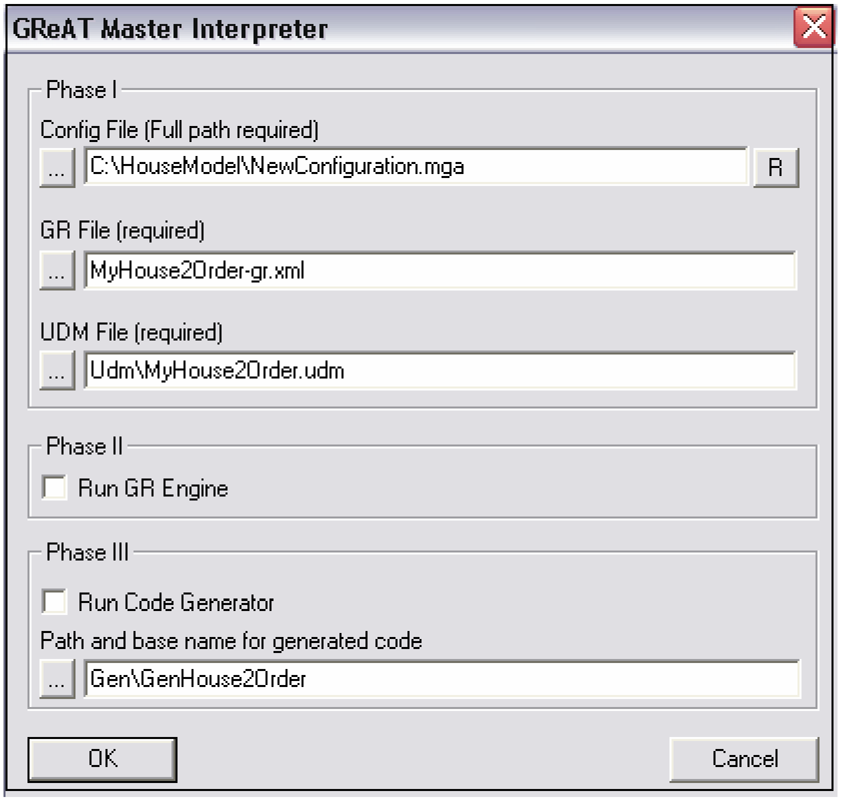
That's it! You are done specifying the transformation.

4.6. Step 6 – Run the GReAT Master Interpreter

No matter which method you use to execute your transformation, you must first invoke the GReAT Master Interpreter to generate the necessary intermediate files used to perform the transformation. These include the domain-specific UDM files, the Configuration file, and the rewriting rules in an internal format used by the GR-Engine.

To run the GReAT Master interpreter, do the following:

a. Go to “File -> Run Interpreter -> GReAT Master Interpreter”
b. A dialog box that looks like this will appear:



Config File (Full path required): C:\HouseModel\NewConfiguration.mga

GR File (required): MyHouse2Order-gr.xml

UDM File (required): Udm\MyHouse2Order.udm

☐ Run GR Engine

☐ Run Code Generator

Path and base name for generated code: Gen\GenHouse2Order

Figure 59 GReAT Master Interpreter Dialog

c. We are interested in Phase I. Accept the default paths, and click OK.
d. The Configuration file, GR file, and UDM files will be generated.

The transformation can now be executed in one of three ways:
1. Using the GR-Engine (see section 5, “The GR Engine”)
2. Using the Code Generator (see section 6, “The Code Generator”)
3. Using the GR-Debugger (see section 7, “The Debugger”)

Important note: if you change your transformation, you must re-run the GReAT Master Interpreter to re-generate the internal format files.

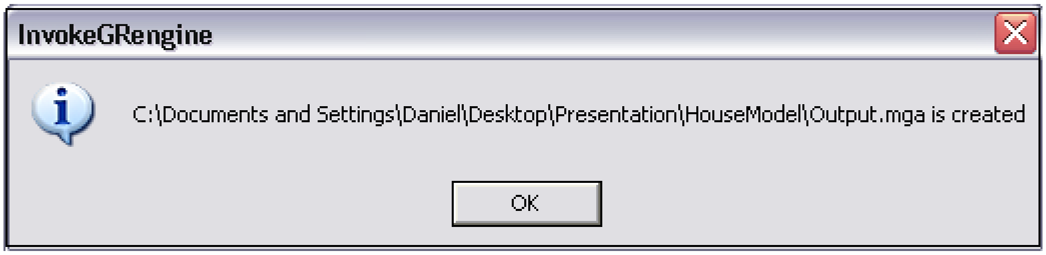
5. The GR Engine

The GR Engine can be used to execute the transformation from within GME. It executes slower than the code generator, but is very useful when creating a new transformation that we change often and want to test quickly.

5.1. Invoking the GR Engine

1. If you aren't already done so, run the GReAT Master Interpreter as described in Section 4.6.
2. Go to "File -> Run Interpreter -> Invoke Graph Rewrite Engine"
3. A dialog box entitled, "GReAT Configuration Dialog" will appear. Before clicking OK, make sure that you have an input model named "HouseInput.mga" in the same directory as your transformation (this was the input file that you specified earlier in the configuration).
4. Click OK.

If the transformation is successful, a dialog box will appear telling you the results of the transformation; in our case, it should look something like the following:



C:\Documents and Settings\Daniel\Desktop\Presentation\HouseModel\Output.mga is created

Figure 60

Congratulations! You've just completed your first GReAT Transformation (no pun intended)! ☺

6. The Code Generator

The code generator can be used to generate C++ files and a Visual Studio Project that can be compiled and used to execute our transformation much faster than using the GR-Engine (approximately 10x-100x faster in most cases). However, the code generator has the disadvantage that it requires visual studio in order to compile the code, and cannot operate within GME.

6.1. Invoking the Code Generator

To invoke the code generator, follow these steps:

1. If you have not already done so, run the GReAT Master Interpreter as described in Section 4.6.
2. Invoke the Code Generator Interpreter. This can be done one of two ways:
   a. File -> Run Interpreter -> Generate Code, OR
   b. Click on the Interpreter Icon, which is shown below:

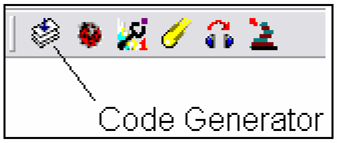
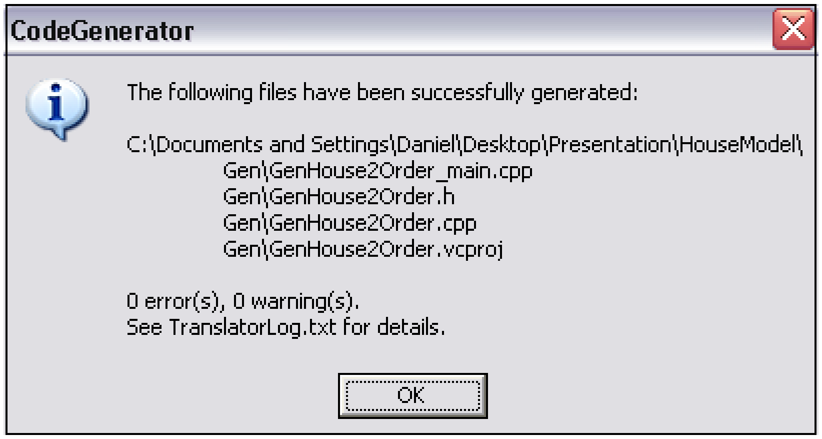


Figure 61 Code Generator Interpreter Icon

If the interpreter ran successfully, you will see a dialog box like the following appear:



```
The following files have been successfully generated:

C:\Documents and Settings\Daniel\Desktop\Presentation\HouseModel\
        Gen\GenHouse2Order_main.cpp
        Gen\GenHouse2Order.h
        Gen\GenHouse2Order.cpp
        Gen\GenHouse2Order.vcproj

0 error(s), 0 warning(s).
See TranslatorLog.txt for details.
```

Figure 62 Dialog box for successful code generation

The code generator generates the appropriate (Visual Studio 6 or .NET depending on the version of GReAT you have installed) C++ Project file that will contain the necessary project settings and files to compile your transformation.

6.2. Compiling the generated code

1. Open the generated project file (in our case, “GenHouse2Order.vcproj”).
3. Build the solution (if a dialog box appears prompting you for a filename for the solution file, accept the default value and click “Save”).

The build should run without any problems, and the executable file “GenHouse2Order.exe” should be created in the \Gen directory.

6.3. *Running the generated executable*

1. Copy the generated executable (“GenHouse2Order.exe”) to the same directory as your GReAT transformation.
2. Open a command prompt, and go to the directory which contains your GReAT transformation (this directory should also now contain the executable “GenHouse2Order.exe” that was copied there in step 4 above).
4. To perform your transformation using the generated .exe file, type “GenHouse2Order –d”. This executes the transformation with the default input files that you specified in your configuration file (“HouseInput.mga” and “Output.mga”).
5. A message should appear saying, “Executing with default arguments.”
6. The transformation should complete very quickly! You can view the output by opening your output file (“Output.mga”).

Congratulations! You have just used the code-generator to run a transformation.

6.4. *Running the generated code with different input files*

Above, we ran our executable with the default input and output files by giving the “-d” command line argument. Now let’s say that we wanted to use a House input model named “MyHouseInput.mga” and we wanted to create an output file named “MyOutput.mga”. All we need to do is run our executable with the following arguments:

```
GenHouse2Order HouseFileType=”MyHouseInput.mga”
OrderFileType=”MyOutput.mga”
```

7. The Debugger

GReAT also comes with a debugger, similar to debuggers used in textual programming languages. The GReAT debugger allows you to set breakpoints in your transformation, and step through the transformation rule-by-rule, observing the values of objects and matches.

7.1. *Invoking the Debugger*

The debugger can be invoked in two ways, either as an interpreter within GME, or from the command line.

7.1.1. Invoking the debugger from the command line

1. If you have not already done so, run the GReAT Master Interpreter as described in Section 4.6.
2. Open a command prompt.
3. Type in grd (the directory from which you type this shouldn't matter, since the GReAT tools are added to your PATH environment variable)
4. The GReAT debugger should appear in a new window.
5. You can load your transformation by going to File -> Open, and then navigating to the directory where your transformation is saved. At this point, open the generated configuration file, named "NewConfiguration.mga".
6. You can now skip to section 7.2 below to read about how to use the debugger.

7.1.2. Invoking the debugger from within GME

1. Click the debugger icon, shown below.

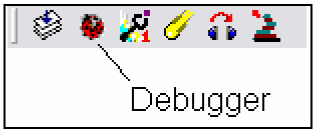
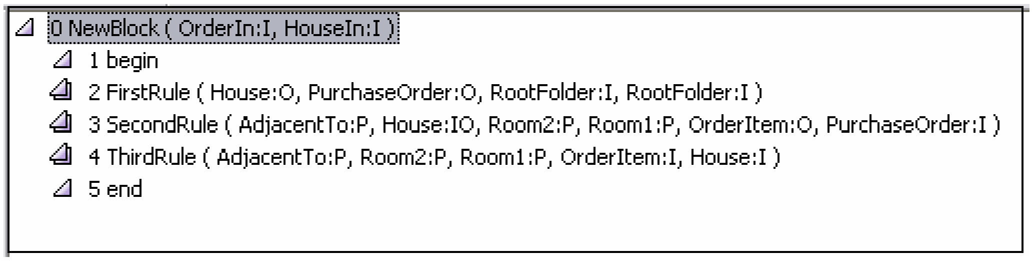


Figure 63 Debugger Interpreter Icon

2. You can also invoke the debugger by going to File->Run Interpreter->Invoke GR Debugger

7.2. Using the debugger

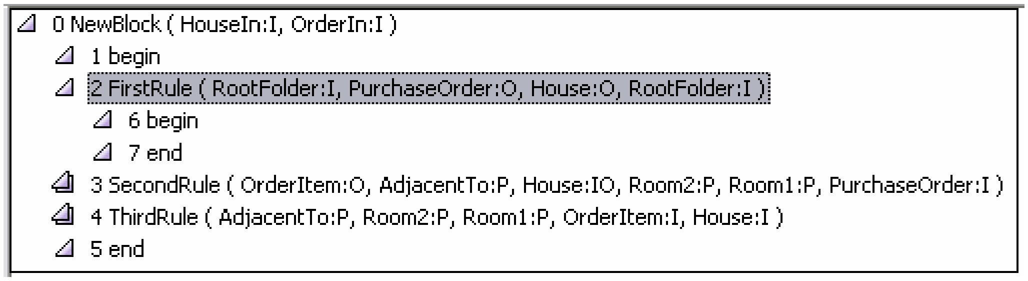
The debugger should now be open, and the main window that displays the names of the rules of your transformation should look like this:



0 NewBlock (OrderIn:I, HouseIn:I)
  1 begin
  2 FirstRule (House:O, PurchaseOrder:O, RootFolder:I, RootFolder:I)
  3 SecondRule (AdjacentTo:P, House:IO, Room2:P, Room1:P, OrderItem:O, PurchaseOrder:I)
  4 ThirdRule (AdjacentTo:P, Room2:P, Room1:P, OrderItem:I, House:I)
  5 end

Figure 64 Rule Window of the Debugger

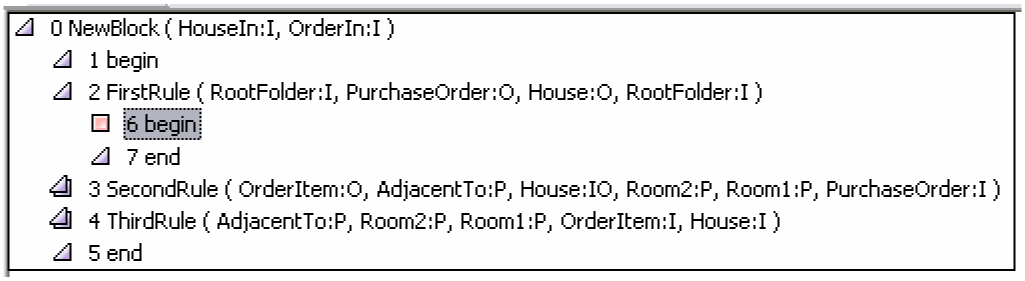
Expand FirstRule by double-clicking on it. The Rule Window should now look like this:



0 NewBlock (HouseIn:I, OrderIn:I)
  1 begin
  2 FirstRule (RootFolder:I, PurchaseOrder:O, House:O, RootFolder:I)
    6 begin
    7 end
  3 SecondRule (OrderItem:O, AdjacentTo:P, House:IO, Room2:P, Room1:P, PurchaseOrder:I)
  4 ThirdRule (AdjacentTo:P, Room2:P, Room1:P, OrderItem:I, House:I)
  5 end

Figure 65 Rule Window after Expanding FirstRule

Now set a breakpoint at begin, which is directly under FirstRule, by highlighting begin (click on it with the left mouse button) and then pressing F9. The rule window should now look like this:



0 NewBlock (HouseIn:I, OrderIn:I)
  1 begin
  2 FirstRule (RootFolder:I, PurchaseOrder:O, House:O, RootFolder:I)
    6 begin
    7 end
  3 SecondRule (OrderItem:O, AdjacentTo:P, House:IO, Room2:P, Room1:P, PurchaseOrder:I)
  4 ThirdRule (AdjacentTo:P, Room2:P, Room1:P, OrderItem:I, House:I)
  5 end

Figure 66 Rule Window after Setting a Breakpoint

Pressing F9 again will remove the breakpoint. You can also use the icons at the top of the debugger to set/remove breakpoints, remove all breakpoints, etc. Simply hold the mouse over any of the icons (shown below) and a caption will appear telling you what that icon does.

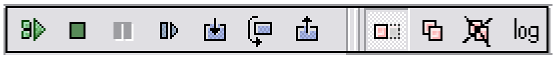
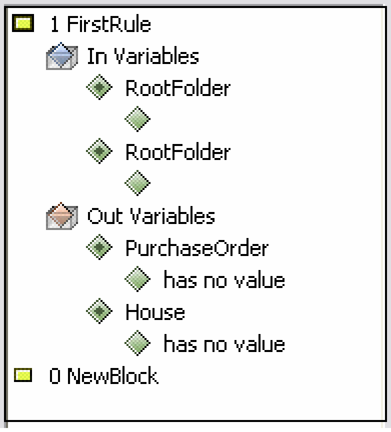


Figure 67 The Debugger Icons

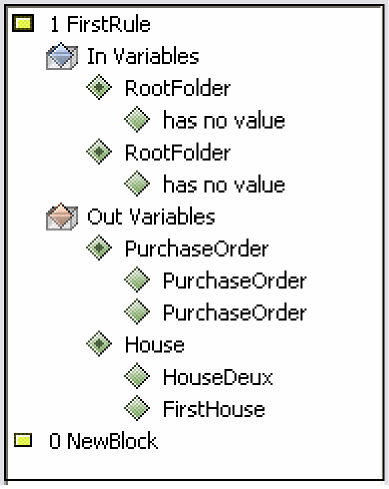
Now let’s run the transformation. Press F5 to begin the program execution. The call stack on the left should look like the following when our breakpoint is reached:



```
1 FirstRule
    In Variables
        RootFolder
            
        RootFolder
            
    Out Variables
        PurchaseOrder
            has no value
        House
            has no value
0 NewBlock
```

Figure 68 Call Stack when Breakpoint is Reached

At this point, the call stack is telling us that there are two variables that input into this rule (In Variables), and there are two variables that are output variables (Out Variables). Because we are at the beginning of this rule's execution, the Out Variables have no value; that is, there has been no matching of our pattern yet, we have simply bound our input variables. Now hit F10. The call stack should now look something like the following:



```
1 FirstRule
    In Variables
        RootFolder
            has no value
        RootFolder
            has no value
    Out Variables
        PurchaseOrder
            PurchaseOrder
            PurchaseOrder
        House
            HouseDeux
            FirstHouse
0 NewBlock
```

Figure 69 Call Stack after FirstRule has completed execution

The call stack now says that the In Variables have no value, and that the Out Variables do have values. This is because now that the pattern matching and object creation has been completed, the bound variables (the two incoming root folders) are no longer in context, and the Out Variables have all been created. We see that there were 2 purchase Orders created, and that there were 2 Houses found in our input model (one house is named FirstHouse, and the other house is named HouseDeux).

We can now either continue stepping through into the rules (F11), step over the rules (F10), or run the program until the next breakpoint is reached (F5). Let's press F5 since we don't have any other breakpoints. The program should finish normally.

Congratulations! You have now used the GReAT Debugger.