1 Resolving Schematic Conflicts

last updated: November 19, 2002 9:13 am

Information conflicts occur when two or more parties:

- Use different syntax to represent the same information
- Use different structures to represent the same information
- Do not recognize different representations of the same information
- Do not agree on common representations of the same information

The Modulant Contextia Interoperability Platform can solve the following kinds of information conflicts:

- *Syntactic conflicts* refer to differences in the format of data values.
- *Schematic conflicts* refer to differences in data structures across applications.
- *Semantic conflicts* refer to differences in the meaning of data.
- *Contextual conflicts* refer to differences in the factors that determine the meaning of data.

When more than one application must share information as part of an identified business process, an interoperability architect working with these applications is bound to encounter one or more of these types of conflicts. As you solve these problems for different interoperability environments, you will become more familiar with the techniques uses to solve each type of conflict.

This tutorial demonstrates one solution to *aggregation conflicts*, a type of schematic conflict.

This tutorial covers the following topics:

- A Closer Look at Schematic Conflicts
- Introducing the Sample Applications
- Developing the Mapping Strategy
- Completing the Context Maps
- Performing an Interoperability Run

ALPHA

A Closer Look at Schematic Conflicts

Schematic conflicts refer to differences of schema, data, or relationships across application data sets. Interoperability architects encounter schematic conflicts of the following types:

- *Data type*: When different applications use different data types for the same data values.

  For example, one application might use a text string to represent a date while another application uses a built-in **date/time** type.

- *Labeling*: This kind of conflict can show up in two situations:
  - When different labels refer to the same attribute.

    For example, one application can have a data field called **Employee_ID** when another one has a field called **Social_Security_Number** for the same purpose.
  - When the same label identifies different attributes.

    For example, two applications can have data fields called **Order_Date**, but in one application it refers to the date an order was placed and in the other one it refers to the date an order was received.

- *Aggregation and structure*: When applications use different data structures to represent equivalent information.

  This is the kind of conflict you will learn to solve in this tutorial.

Introducing the Sample Applications

Figure 1 shows an example of an aggregation conflict between two application data structures. For simplicity, this example shows the same data values in each application's format; in a real interoperability environment, of course, each application would have different data values.

In this example, the architects of each application have made different design choices. The designers of Application 1 encoded information about both cars and trucks in a single logical entity. In contrast, the designers of Application 2 chose to store information about cars separately from information about trucks. More specifically:

- In Application 1, the type of vehicle is explicit, in the value of the **type** field.
- In Application 2, the type of vehicle is implicit, only available in the name of each logical entity.

ALPHA

Figure 1: Example of an Aggregation Conflict

Application 1:

AUTOMOBILE	
NAME	TYPE
Camry	Car
Pathfinder	Truck
Saab 900	Car
Avalanche	Truck

Cars and trucks are aggregated in a single entity with a **type** attribute to distinguish them.

Application 2:

CAR
NAME
Camry
Saab 900

TRUCK
NAME
Avalanche
Pathfinder

Cars and trucks are grouped in two different entities—no **type** attribute is needed.

In an interoperability environment comprising these two sample applications, consider that Application 1 contains data needed by Application 2. This places Application 1 in the role of *information provider* and Application 2 in the role of *information consumer*.

To solve the aggregation conflict, you need two *context maps*, one for each application. A context map describes the logical structure of a set of application data, along with the context of the data and the relationships among the data elements.

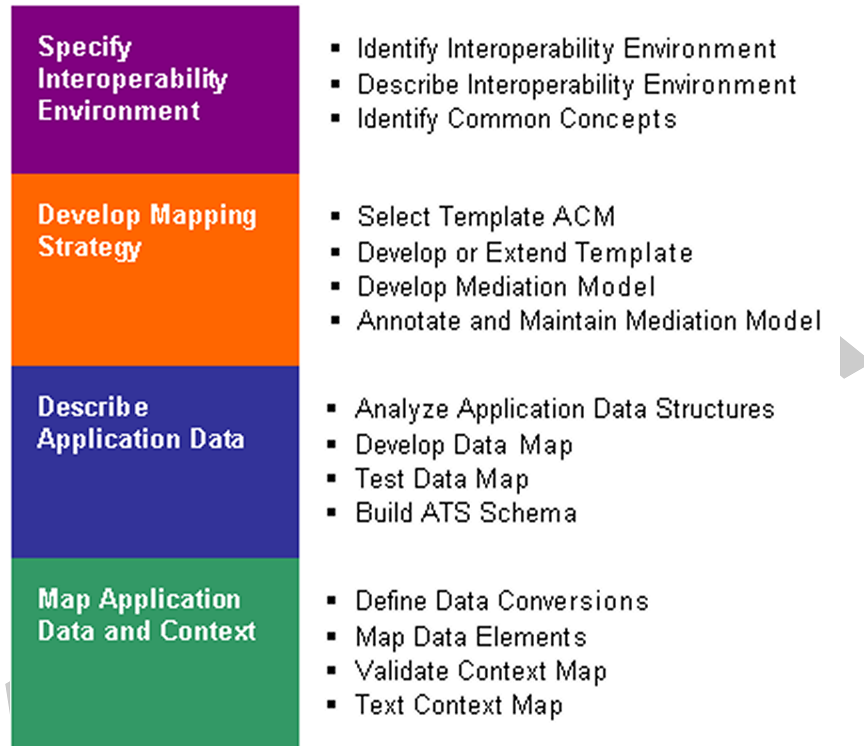
The next section shows the mapping strategy that you will use to create these context maps.

ALPHA

Developing the Mapping Strategy

Figure 2 shows an overview of the thought process involved in solving interoperability problems. Each stage corresponds to one of the methods in the Modulant methodology, known as the *Context-based Information Interoperability Methodology* (CIIM, pronounced *simm*).

Figure 2: CIIM Methods and Procedures



In this demo scenario, you will only create context maps; everything else you need has been provided for you.

A mapping strategy defines how you, as an interoperability architect, will map data elements from each application to an abstract representation, known as an *Abstract Conceptual Model* (ACM). The abstract representation acts as a mediator between the different application structures and contexts.

Developing a mapping strategy includes the following steps:

- Analyzing Sample Data
- Identifying Common Concepts

Analyzing Sample Data

The first step in developing a mapping strategy is to identify the meanings of the data from each application. To do this, you examine actual data from each application, in collaboration with *domain experts* for each application, who are familiar with how those applications are used.

For the example in Figure 1, sample application data in XML looks like this (using the same data values for both applications to allow easy comparison of the data structures):

Application 1:

```
<autodb>
   <car code="RF234" type="car">
      <name>toyota camry</name>
   </car>
   <car code="RF235" type="car">
      <name>saab 900</name>
   </car>
   <car code="RF6735" type="truck">
      <name>nissan pathfinder</name>
   </car>
   <car code="RF784" type="truck">
      <name>chevy avalanche</name>
   </car>
</autodb>
```

Application 2:

```
<autodb>
   <cars>
      <car code="RF234" name="toyota
         camry"/>
      <car code="RF235" name="saab 900"/>
   </cars>
   <trucks>
      <truck code="RF784" name="chevy
         avalanche"/>
   <truck code="RF6735" name="nissan
         pathfinder"/>
   </trucks>
</autodb>
```

ALPHA

Notice that both applications describe both cars and trucks. In Application 1 the type of vehicle is the value of the **type** attribute of the XML element **car**. In Application 2 the type of vehicle shows only in the name of the XML element that contains the data values, but not in the data values themselves. The first thing to notice here is the actual data values used by Application 1, **car** and **truck**. You will use these values, spelled exactly the way Application 1 spells them, as meta-data in the context map for Application 2.

This brings up a critical point in designing interoperability solutions: neither of the domain experts for these applications needs to know about how the other application represents specific data values. But, in your role of interoperability architect, you must be aware of this information to create a mediation layer that will enable the Modulant Contextia Interoperability Server to accommodate the context of all of the applications in the interoperability environment.

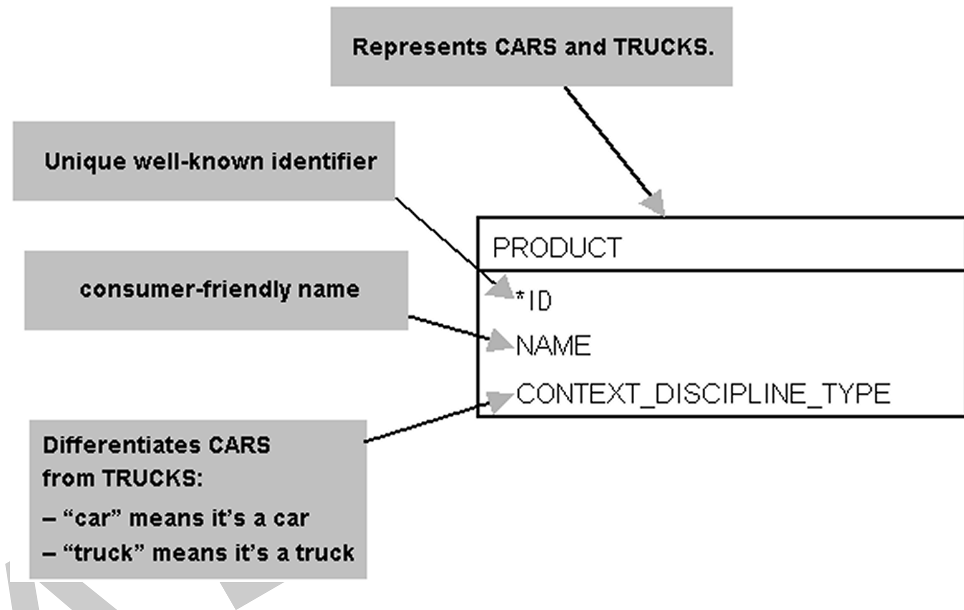
So, in discussions with domain experts for each application, you learn that both applications store information about cars and trucks, specifically, an identifier (the code), the model of the vehicle (the name), and whether it is a car or a truck (the type).

Identifying Common Concepts

Once you understand the data values in each application, you are ready to identify the common concepts that these applications share. To do this, you examine the high-level concepts available in your Abstract Conceptual Model. The Modulant ACM contains concepts such as **class**, **person**, **product**, and **organization**. In this case, cars and trucks can all be considered as products.

Figure 3 shows the high-level mapping strategy for this interoperability environment. The **product** entity in the Modulant ACM has attributes that represent a unique identifier, the product name, and the type of product.

Figure 3: Basic Mapping Strategy for Application 1 and Application 1

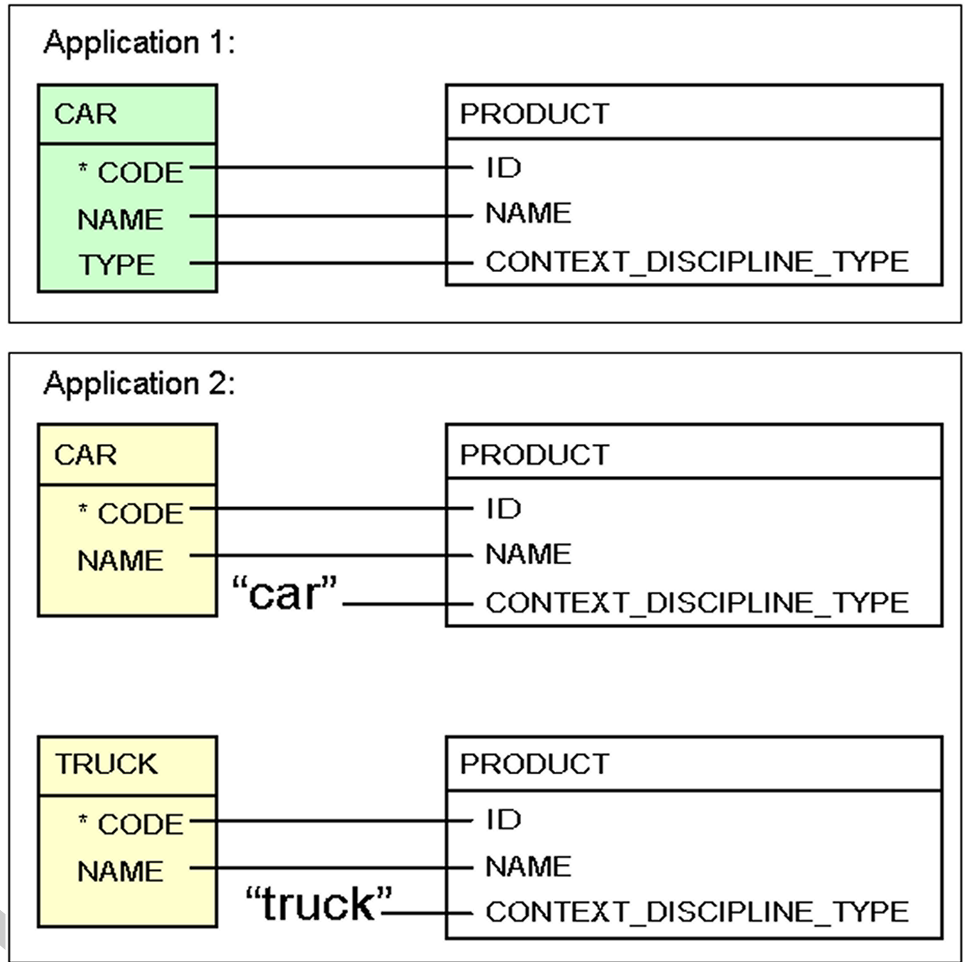


To solve the aggregation conflict, you use each application's context map to specify whether a vehicle is a car or a truck:

- For Application 1, by using the value of the **type** field.
- For Application 2, by supplying this information as meta-data in the form of context strings.

Using the high-level strategy, Figure 4 shows the correspondences between the data elements in each application and the attributes of the abstract **product** entity.

Figure 4: Detailed Mapping Strategy for Application 1 and Application 1



Completing the Context Maps

Now that you have identified the basic concepts represented by the application data in this interoperability environment, you are ready to create the context maps. You will add mapping statements and context maps to create two context maps, which you can use to perform an interoperability run.

Completing the context maps involves:

- Starting the Context Mapper
- Completing the Context Map for Application 1
- Completing the Context Map for Application 2

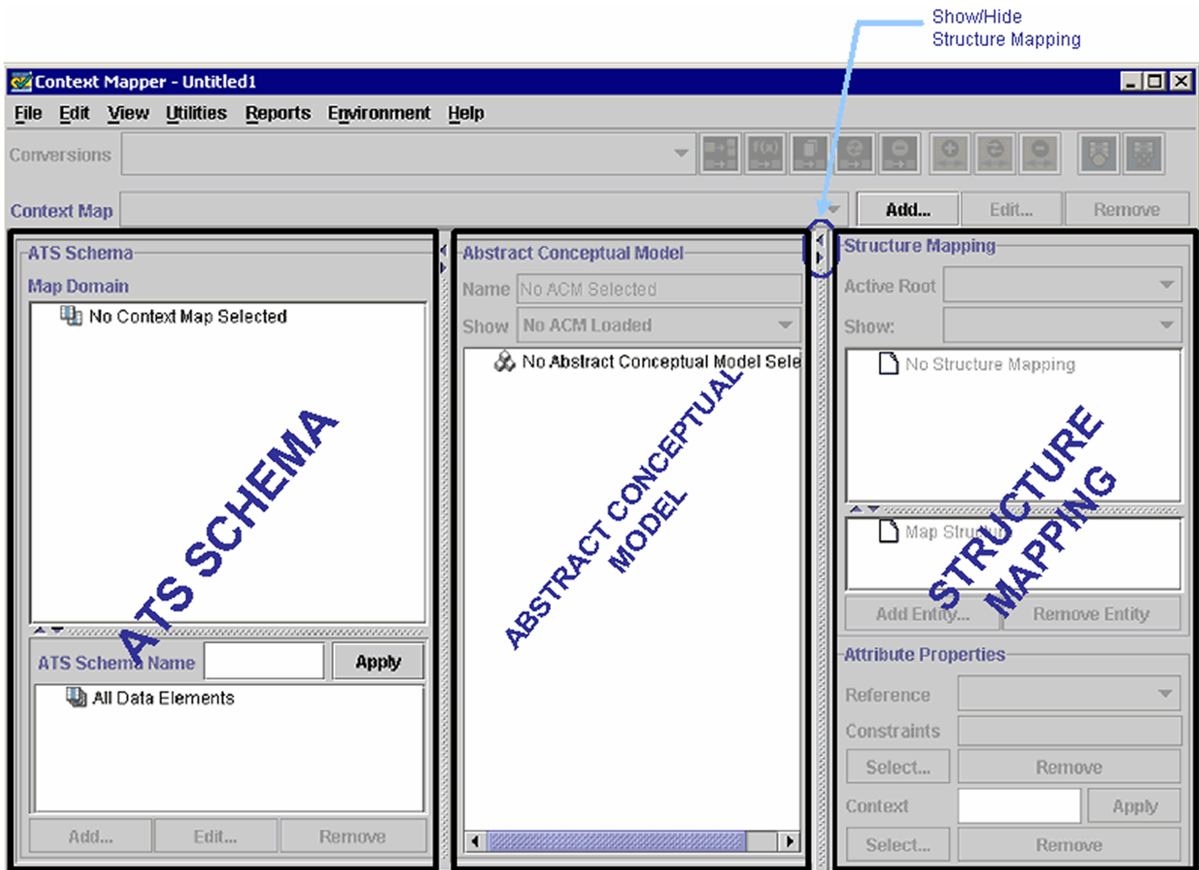
Starting the Context Mapper

The Contextia Context Mapper enables you to define one or more context maps for an application schema. The main window shows you the data elements in your application along with the parts of the context map as you develop it. In addition, you can view all or part of the structure of the Abstract Conceptual Model that contains the mapping targets for your data elements.

To start the Contextia Context Mapper, do one of the following:

- From the Windows desktop, select **Start > Programs > Modulant Contextia Workbench > Contextia Contextia Context Mapper**.
- On a UNIX system, go to the **bin** subdirectory of your Workbench installation directory, and run the shell script called **cxmapper**.

The Contextia Context Mapper starts in its own window:



The main window of the Context Mapper has three regions:

- The **ATS Schema** region lists the data elements in your application and the data elements that are part of the current context map. In this region, you can view and change the properties of data elements and add data elements to the domain of a context map.
- The **Abstract Conceptual Model** region lists the entities in the abstract representation that contains the mapping targets for the data elements in your application.
- The **Structure Mapping** region displays the structure mapping for a context map. The structure mapping is a set of instances of Abstract Conceptual Model entities and their relationships that together provide the context for that data elements that appear in the context map.

Completing the Context Map for Application 1

The mapping process expects a *root data element* for each logical entity. The root data element is one that connects a set of other data elements, usually the unique identifier of a logical entity. Root data elements serve as anchor points for mapping related data elements; often they are the primary keys of database tables, or other unique identifiers.

To start your mapping, you specify a root data element and its mapping target. Application 1 has one logical entity, **car**. The root data element is the key field **code**. Remember that the abstract concept that represents cars is **product**.

To complete the context map for Application 1:

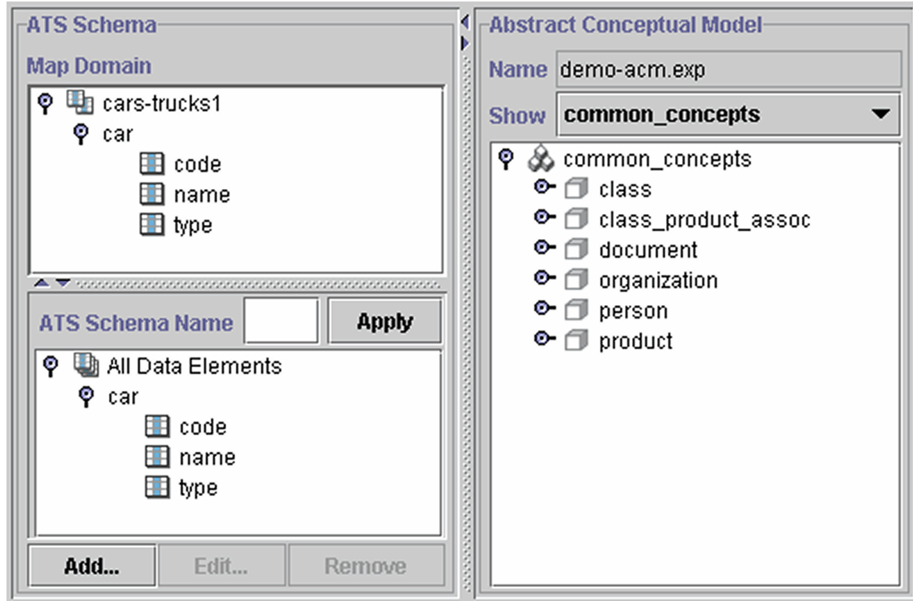
1. Select **File** > **Open**.

   The Context Mapper starts looking in the **cxm** directory of your Workbench installation.

2. In the **demo1** subdirectory, select **cars-trucks1-start.cxm** and click **Open**.

ALPHA

The **ATS Schema** region shows the data elements in this context map (in the **Map Domain** section) and the **Abstract Conceptual Model** region shows the common concepts you can use for mapping:



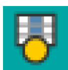
3 In the **Map Domain** section, expand the logical entity **car** to see its attributes.

Tip: To expand or collapse a node, click the icon to the left of its name.

4 To specify **code** as the root data element:

a In the **Map Domain** section, select **code**.

Notice that several of the toolbar buttons on the toolbars are now active.



b On the mapping toolbar, click the **Set Root** button.

The icon next to **code** changes to show that it has been identified as a root data element.

5 To map the data element **code**:

a In the **Map Domain** section, select **code** if it is not already selected.

b In the **Abstract Conceptual Model** region, scroll to the **product** entity and expand it.

c Select the **id** attribute of product.

d On the mapping toolbar, click the **Create Mapping Statement** button.

ALPHA

The **Finish Mapping Statement** dialog box appears:

Finish Mapping Statement
Target Entity: product
Direction: Input/Output
Entity Usage: New Entity Usage
Entities Referenced By Target
OK | Cancel

e Next to **Direction**, accept the default value **Input/Output**.

f Next to **Entity Usage**, accept the default value **New Entity Usage**.

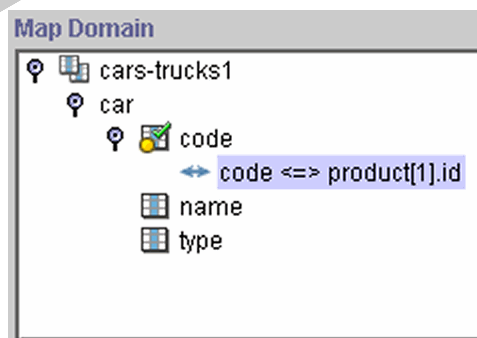
The *entity usage* identifies a single occurrence of an ACM entity in a structure mapping. Because the structure mapping is empty at this point, you are adding a new occurrence of the **product** entity.

g Click **OK**.

The Context Mapper creates a mapping statement specifying **product.id** as the mapping target for **code**.

Note: Names of elements of the Abstract Conceptual Model use the notation ***entity_name.attribute_name***; for example, **product.id**.

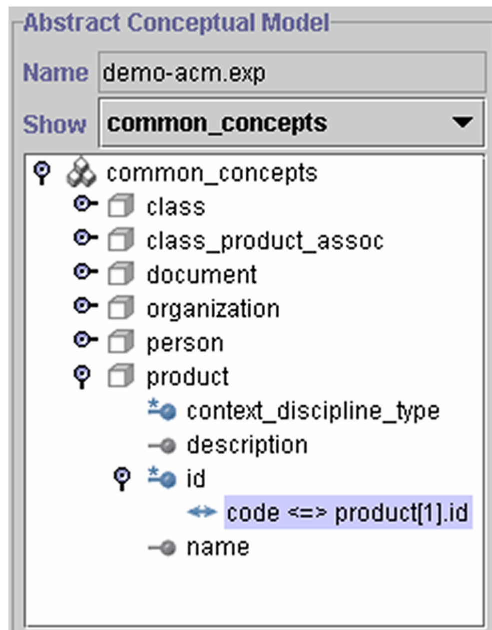
In the **Map Domain** section, you can see the new mapping statement below the root data element:



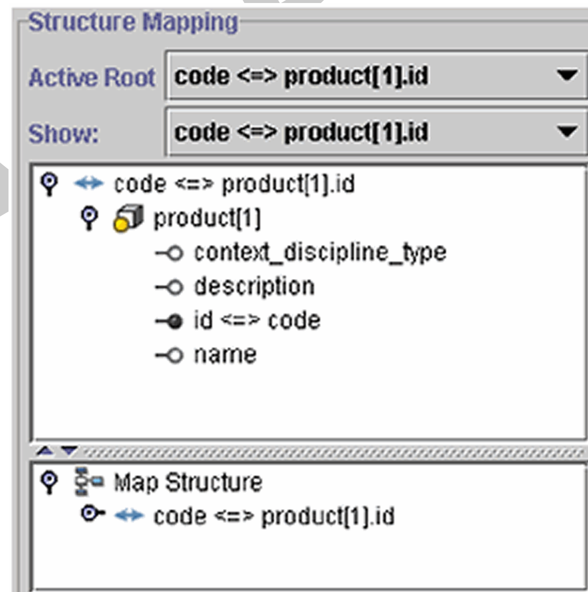
The **product** entity has a number **1** after it. This number is the entity usage, and indicates that this is the first occurrence of **product** in this structure mapping.

ALPHA

The new mapping statement also appears in the **Abstract Conceptual Model** region, below **product.id**, the mapping target you chose:



h In the **Structure Mapping** region, expand the **product** node at the top of the element structure:



ALPHA

In the top pane, the *element structure* shows the context associated with a single mapping statement. In the bottom pane, the *map structure* shows the entire structure mapping for a context map. As you add entities to each element structure, they all appear in the map structure.

6 To map the **name** attribute of **car**:

   a In the **Map Domain** section, select **name**.

      Remember that **name** indicates the name of this car model.

   b Drag **name** all the way to the **Structure Mapping** section, to the **name** attribute under **product[1]**.

      The Context Mapper adds a new mapping statement under **name** in both the **Map Domain** section and the **Abstract Conceptual Model** region, and adds a link from **product.name** to the **name** attribute of **car**.

      That's all you need to do to map **name**.

7 To map the **type** attribute of **car**:

   a In the **Map Domain** section, select **type**.

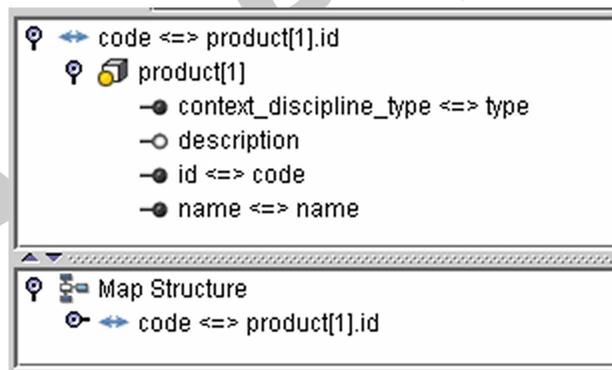
      Remember that **type** identifies whether this is actually a car or a truck.

   b Drag **type** all the way to the **Structure Mapping** section, to the **context_discipline_type** attribute under **product[1]**.

      You can use **context_discipline_type** attribute of **product** to specify what kind of product this is. The **type** attribute of **car** provides this information.

      That's all you need to do to map **type**.

The structure mapping should now look like this:



8 To save your changes, select **File > Save As**.

The Context Mapper starts looking in the **cxm** directory of your Workbench installation.

Note: The file you originally opened, **cars-trucks1-start.cxm** was set to read-only as part of this installation. Modulant recommends that you save your changes in a separate file, so that you or someone else can do this exercise again starting with the same files.

9 Go to the **demo1** subdirectory and type a new name, such as **cars-trucks1.cxm**, and click **Save**.

That's all you have to do for the first context map. Now you're ready for the next one.

Tip: The file **cars-trucks1-complete.cxm** in the **cxm\demo1** subdirectory of your Workbench installation contains a finished copy of the context map you just created.

Completing the Context Map for Application 2

Completing the context map for Application 2 follows the same basic steps as completing the context map for Application 1, with a few differences:

- Application 2 has two logical entities, which you map separately.
- Each logical entity has a root data element.
- Your map structure will have two copies of **product** this time, one for cars and one for trucks.
- For each instance of **product**, you provide the implicit information about the data—whether a data value is a car or a truck— as a context string.

To complete the context map for Application 2:

1 Select **File > Open**.

The Context Mapper starts looking in the **cxm** directory of your Workbench installation.

2 In the **demo1** subdirectory, select **cars-trucks2-start.cxm** and click **Open**.

3 In the **Map Domain** section, expand both **car** and **truck**.

Notice that each logical entity has two attributes, **code** and **name**. In this application, there are no data elements that identify whether a vehicle is a car or a truck. At this point, this information only appears in the names of the entities.

4 To map the data elements in the **car** entity:

   a Following the instructions in step 4 on page 10, set **code** as the root data element of the **car** entity.

   b Following the instructions in step 5 on page 10, map the **code** attribute of **car** to **product[1].id**.

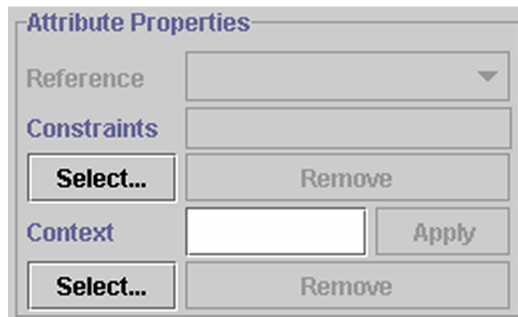
   c In the **Map Domain** section, select the name attribute of **car** and drag it to **product[1].name** in the **Structure Mapping** region.

   Because the data structures for this application contain no explicit indication of the type of vehicle, you must specify this as meta-data in the form of a context string.

   d In the element structure, select **product[1].context_discipline_type**.

ALPHA

The **Context** field in the **Attribute Properties** section at the bottom of the **Structure Mapping** region becomes active:

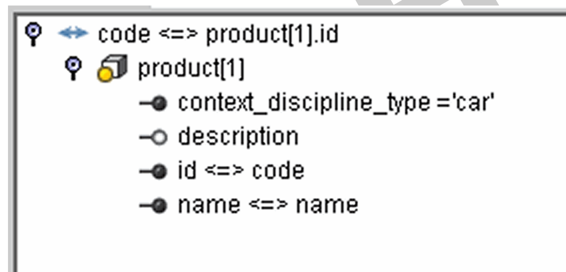


e In the **Context** field, type `car` to specify that all of the data values for this logical entity represent cars.

Remember that the data values in Application 1 for the type attribute of the car entity were **car** and **truck**. To accommodate the context of both applications, you must match the spelling of those data values when you enter context strings.

f Click **Apply**.

The context string you entered appears in the structure mapping:



5 To map the data elements in the **truck** entity:

a Following the instructions in step 4 on page 10, set **code** as the root data element of the **truck** entity.

Because truck is a separate logical entity, you will map its data elements to a new copy of the **product** entity from the ACM: **product[2]**.

b Following the instructions in step 5 on page 10, map the **code** attribute of **truck** to **product[2].id**.

c In the **Map Domain** section, select the name attribute of **car** and drag it to **product[2].name** in the **Structure Mapping** region.

d In the element structure, select **product[2].context_discipline_type**.

e In the **Context** field, type `truck` to specify that all of the data values for this logical entity represent trucks.

f Click **Apply**.

6 To save your changes, select **File > Save As**.

The Context Mapper starts looking in the **cxm** directory of your Workbench installation.

7 Go to the **demo1** subdirectory and type a new name, such as **cars-trucks2.cxm**, and click **Save**.

Tip: The file **cars-trucks2-complete.cxm** in the **cxm\demo1** subdirectory of your Workbench installation contains a finished copy of the context map you just created.

Now you are ready to test the context maps with sample data.

Performing an Interoperability Run

To test your context maps, you will perform an interoperability run with Application 1 as the source and Application 2 the target application. To do this, you will use sample data provided as part of your Workbench installation.

To Technical Reviewers: *Until the Web interface is available, this section uses the high-level design and improvises about what controls will actually be in the final version.*

To perform a test interoperability run:

1 In a browser window, go to <URL>.

The login page appears:

2 Enter the user name and password you used when you downloaded the evaluation version of Modulant Contextia, and click **OK**.

3 On the evaluation home page, click **Upload** to test your mappings.

ALPHA

The upload page appears:

Upload Page
(textbox for
CXM file)

4 Next to the **Source Context Map** field, click **Browse**.

5 In the **demo1** subdirectory, select the file in which you saved the Application 1 context map and click **Open**.

6 Next to the **Target Context Map** field, click **Browse**.

7 In the **demo1** subdirectory, select the file in which you saved the if context map and click **Open**.

8 To start the interoperability run, click **Go**.

The Interoperability Server uses the context maps you provided to transform data in the format used by Application 1 into the format used by Application2.

When the interoperability run is finished, the get results page appears:

Get Results
(link to result
data)

ALPHA

9 To see the source data from Application 1 and the target data created for Application 2, click **View Source and Target Data**.

On the view results page, the XML data should look like this:

Source: Application 1	Target: Application 2

<pre><autodb>
   <car code="RF234" type="car">
      <name>toyota camry</name>
   </car>
   <car code="RF235" type="car">
      <name>saab 900</name>
   </car>
   <car code="RF6735" type="truck">
      <name>nissan pathfinder</name>
   </car>
   <car code="RF784" type="truck">
      <name>chevy avalanche</name>
   </car>
</autodb></pre>
	<pre><autodb>
	   <cars>
	      <car code="RF234" name="toyota
	         camry"/>
	      <car code="RF235" name="saab 900"/>
	   </cars>
	   <trucks>
	      <truck code="RF784" name="chevy
	         avalanche"/>
	   <truck code="RF6735" name="nissan
	         pathfinder"/>
	   </trucks>
	</autodb></pre>

Congratulations! You've just completed a successful interoperability project.

Do you want to try another one?

- Scenario 2: Resolving Semantic Conflicts

ALPHA