Classes 17 and 18

Learning Objectives

- Understand data aspects of workflows
- Be able to model data aspects in YAWL

Readings

This class is the fifth in a series of six classes (13 to 20) which will introduce you to the control-flow, resource, and data-flow perspective of a workflow model. The previous classes covered the control-flow and the resource perspective. In this class, we will define the data perspectives for workflows.

So far, you have seen workflows without data. Admittedly, they were pretty meaningless by themselves, but did show the routing and interaction strategies that workflow systems can provide. In this class, we will finally add some data that flows through the process. For example, when making travel reservations, there will be information about the traveller, the hotel, car, and flight bookings, and the billing and accounting information. All of these need to be entered into the process by the users, checked, edited, approved, etc.

Two notes of advice. First, in a real workflow setting, not all work items are executed by the workflow system itself. In many cases, existing software, such as accounting systems, inventory systems, sales systems, etc. are used to work on each work item. These software systems of course maintain their own data, so that the workflow system does not have to serve as a database but is only concerned with getting data from task to task (and system to system). In YAWL, such software can be integrated either using web-service interfaces and the web-service invocation service about which you have read briefly in Section 7.3.5 when you read Chapter 7 for class 9. You also read an article by Ferris & Farrel (2003) about web-services in class 6. So here is an instance of where this is actually used. An alternative to this is the integration through the codelet service, one of the four sub-services of the resource service. This was also briefly described in Chapter 7, in Section 7.3.3.

To keep things simple for this course, we do not have such extra software. In fact, integrating these systems is often the most difficult task in a workflow management project, because of the different technologies involved. Because we do not have such software, the data of completed workflows remains in the YAWL workflow engine, but cannot be easily retrieved or viewed. Consequently, the focus of this chapter is how data is moved from task to task or work item to work item from the perspective of the workflow management system.

As the previous class, this class also relies on Chapters 2 and 8 of the textbook. In addition, there is a brief reading by Bratosin (2009) that adds a little more background and fills in some details. Data in YAWL is maintained in XML format. As part of the class 6 readings, you read a brief introduction to XML by Treese (1998), so you should have a rough understanding of what it is. The article by Bratosin (2009) explains some more XML bits in the context of using them with YAWL.

This class is heavily hands-on with the YAWL software. As for the previous classes, the YAWL users manual might also be helpful for the detail. If you wish to read the user's manual, the relevant section in there is Section 5.

You may also wish to view the following tutorial videos on the course web-site:

- Video Y3: YAWL – Data Definitions
- Video Y4: YAWL – Data Definitions Demo 2

Chapter 2

At this point you will have read most of chapter 2 already, having covered the control-flow and resource perspective. The remaining parts of chapter 2 that deal with the data-flow perspective and which you should read for this class are the following:

- Section 2.1
- Section 2.2.2
- Section 2.5

The data patterns in Section 2.2.2 are again broken into groups. Data patterns describe the visibility of data (i.e. which data items can a task or work item "see"), the interaction of processes with respect to data, the transfer of data from workflow engine to work items and back again, and the way in which data can be used to make routing decisions.

The data visibility patterns are very general. In fact, they are so general that YAWL only supports three of them (see Table 2.6 on page 70). Data visibility concerns what data is visible (i.e. can the data be viewed or edited) from which task or work item. Obviously, if a data item is defined local to a task, it is visible. However, data items may also be defined at other levels and it is important to understand if they are visible. Other workflow management systems may provide support for additional patterns, so that it is important to understand the possible data visibility patterns.

Data interaction is a bit of a misnomer, as it's really about the interaction of process components with respect to the data; the data itself does not interact. This specifies for example how data is passed from one task to another (there are multiple ways in which this could be done), or how data is passed from a task to its sub-tasks. Again, there are multiple ways of doing this and the chapter has some figures to make it easier to understand (Figure 2.9 to 2.11).

Data transfer patterns describe how data is transferred between two work items or tasks. Of the four patterns described here, YAWL only supports the first one.

Data based routing patterns describe the ways in which data may be used to decide the routing of work items along the process. For example, when you have modelled an XOR-split ("either A or B, but not both"), you may wish to specify when to do A and when to do B, based on the value of some data in the process. For example, when the value of a purchase order is greater than $1000 it may need to be approved by a manager. If not, it can be purchased without approval.

Once you understand the principles from Section 2.2.2 and the idea of XML from the earlier Treese (1998) article, you should be able to make sense of Section 2.5 which describes the data features provided by YAWL. In this section, the idea of a query is central, so I will explain this a bit.

In YAWL, the data associated with a workflow instance is stored in Net variables. The data that is visible in each task is stored in task variables. Hence, at the beginning of a task, we need to select the

relevant/necessary parts of the net data and put them into task variables. At the end of the task, we need to select the task data (which may have changed as part of the task execution) and put it back into the corresponding net variables. The two data transfers correspond to the specification input parameters (and input query) and output parameters (and output query) in YAWL. A query is just a way of selecting a set of data. So, the input query selects data from net variables and puts them into task variables, while the output query selects data from task variables and puts them into net variables. When there is a task variable for every net variable, the queries are very simple. It becomes a bit more complicated when data has to be transformed, combined, split, etc. as part of the transfer process.

Because queries in YAWL need to select XML data, they are written in the XML Query language Xquery. The Bratosin (2009) article has some details and further pointers in Section 3 of that article.

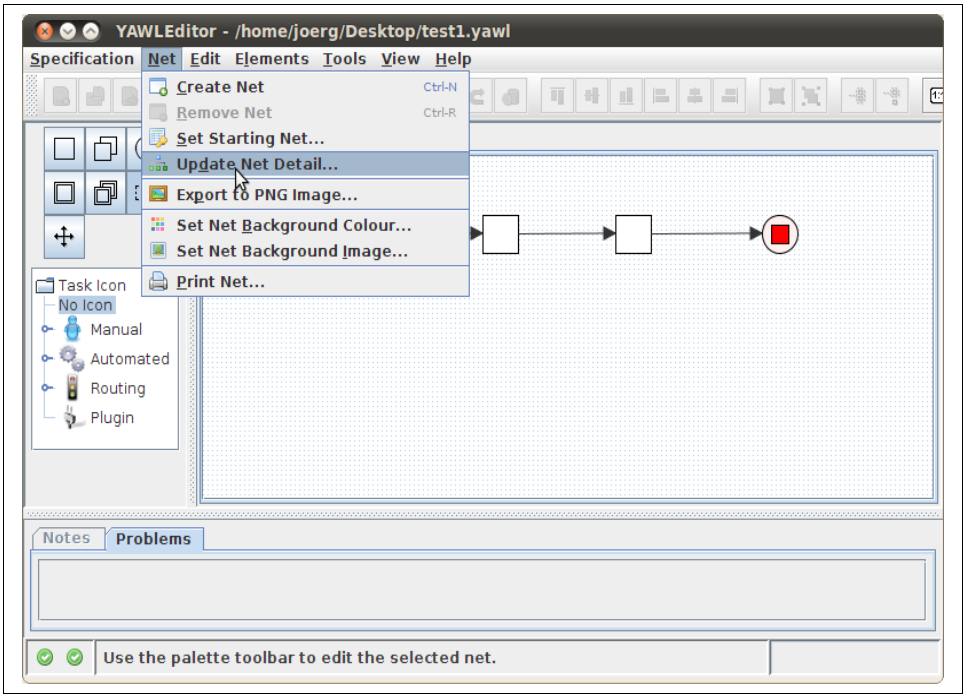
Chapter 8

Chapter 8 deals with the specifics of defining data in the YAWL editor. For this class, you should read the following sections:

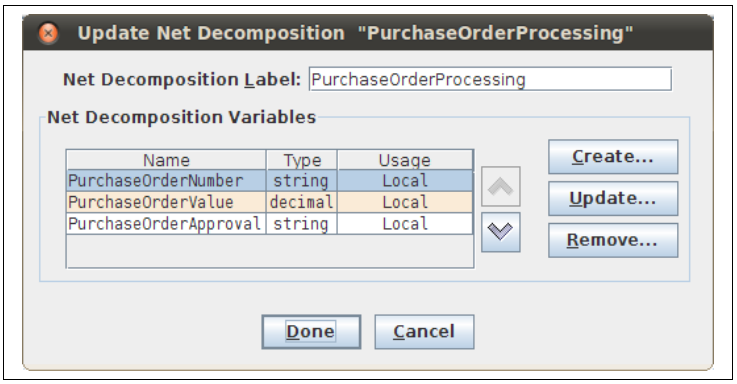
- Section 8.3
- Section 8.2 (re-read to cover data-based routing)

To use data in a workflow, you will need to create net variables which contain the relevant data. You will then need to decompose each task. The easiest way to decompose a task is to a direct data transfer. Recall that you did this already in the previous class to assign resources. Here we will use data transfer decompositions to add input and output parameters to tasks.

To create a net variable, choose “Update net details ... “ from the “Net” menu in the YAWL editor. You will then see a dialog box in which you can create, update and remove net variables and their data-types. You can also provide a name to the net decomposition (the name of the net).

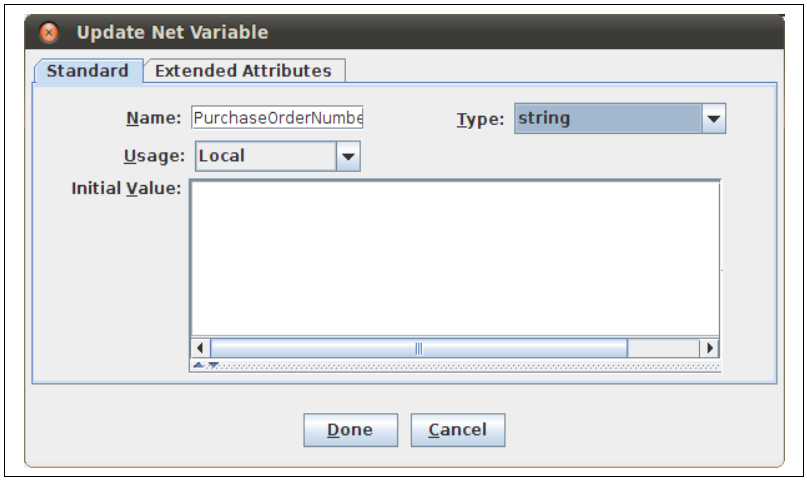


Menu entry to update net variables



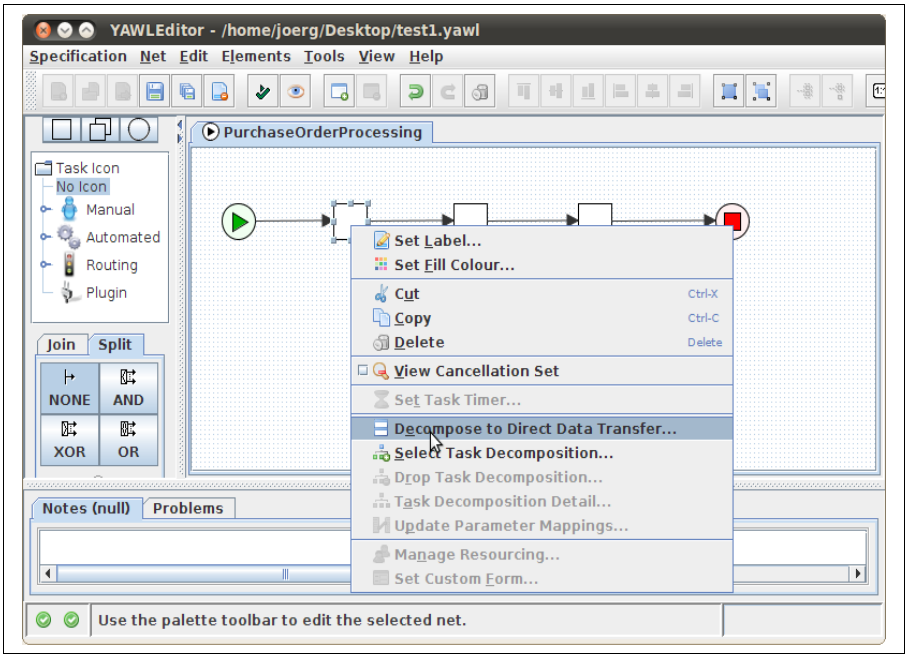
Updating net decomposition and net variables

Each variable has a name and a data-type which you can select form the drop-down box on the right. Simple data types are already provided in YAWL but if you need to make application specific data types, e.g. for purchase orders, sales invoices, etc. see Section 2 in the article by Bratosin (2009). For usage, you should select "Local" for net-level variables.



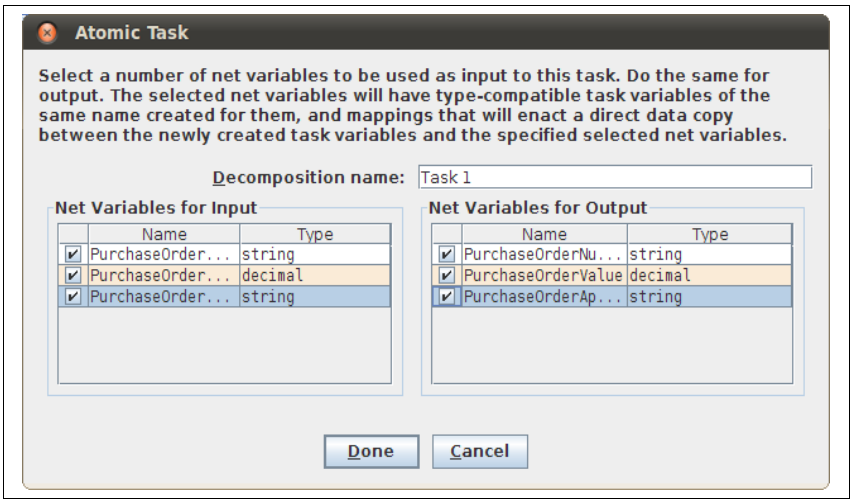
Creating or updating a net variable

Once you have created net variables, you can, for each task, create a task decomposition (typically a direct-data transfer decomposition, which is easiest) by right-clicking on each task and selecting "Decompose to direct-data transfer" from the menu:



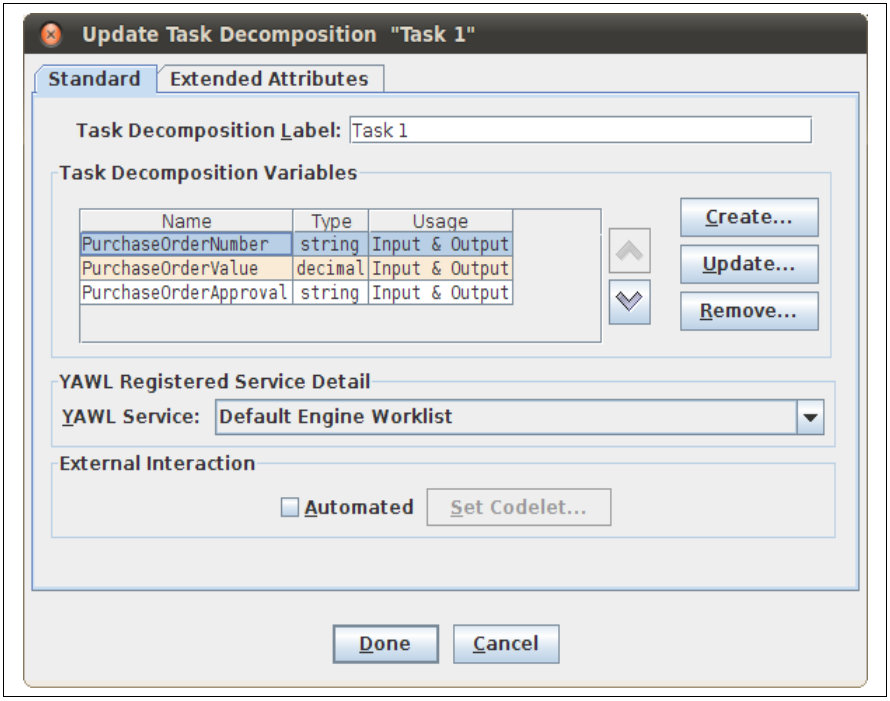
Selecting direct data transfer decompositions

You have seen the following dialog box in the previous class, but because we had not specified any data, it was empty then. Now the input parameter and output parameter lists are filled with the net variables you have just defined. You can select the ones that you want to import to the task (which are then available in the task) and which ones you want to output from the task (whose data is written back to the net level when the task completes). You can also provide a name for the task decomposition (i.e. the name of the task).



Specifying input and output parameters for a direct data transfer decomposition

You are now done with specifying the data perspective (in this very simple way). To check that everything is ok, you can right-click on the task and select “Update task decomposition”. You will see a dialog box that shows you the task variables that were automatically created.



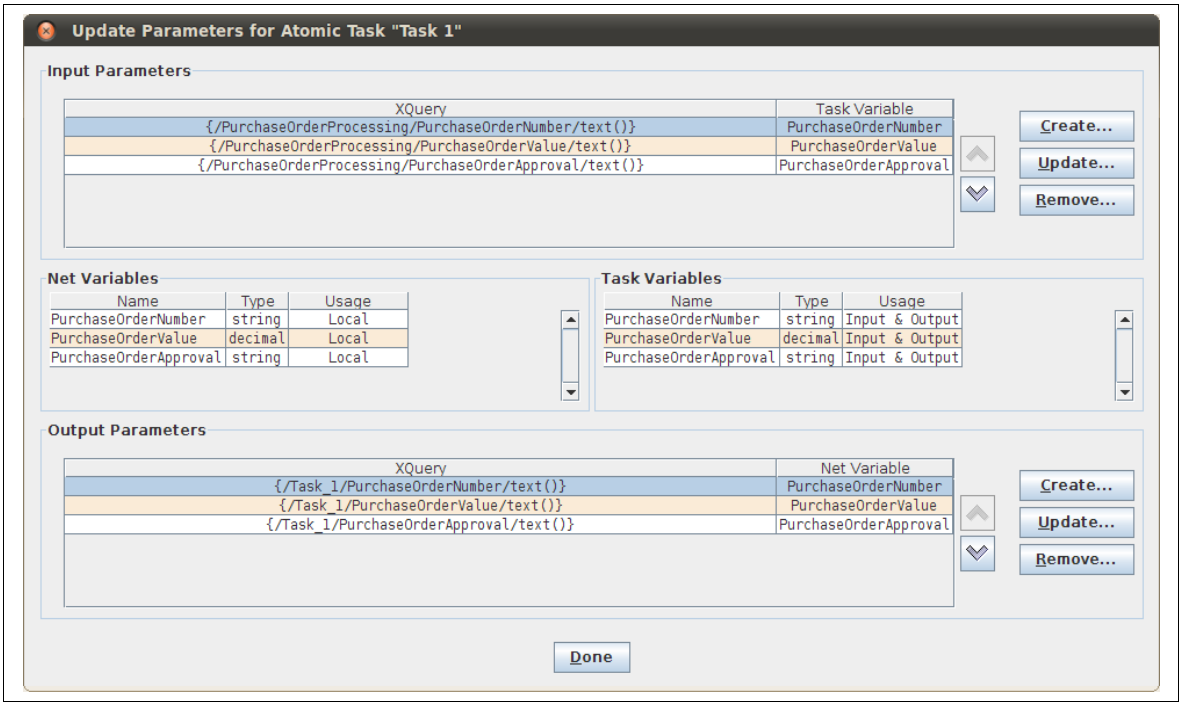
Updating the task decomposition

In this case, there are three task variables, because I had chosen to input and/or output each of the three net variables. Because I had chosen input and output for all three, the task variables are all classified as Input/Ouput variables.

If you wish to make changes to the input and output parameters and the input and output queries, you can right-click on a task and select “Update parameter mappings”. The following dialog box will show you the input/output mappings and the automatically created XQueries. You can now make changes to any or all of these. In the following screenshot you can see that each of the task variables has an associated XQuery. Each of these XQuery expressions selects some data from the net variables and puts it in the specified task variable. The XQuery expressions for the output parameters work the other way around. For each net variable, there is an associated XQuery, which selects some data from the task variables and puts it in the specified net variable when the task finishes.

Note that when you specify a decomposition to direct data transfer, the net and task variables have the same name. However, you can tell them apart, because the net variables are of the form `/net_name/variable_name` while the task variables are of the formal `/task_name/variable_name.`

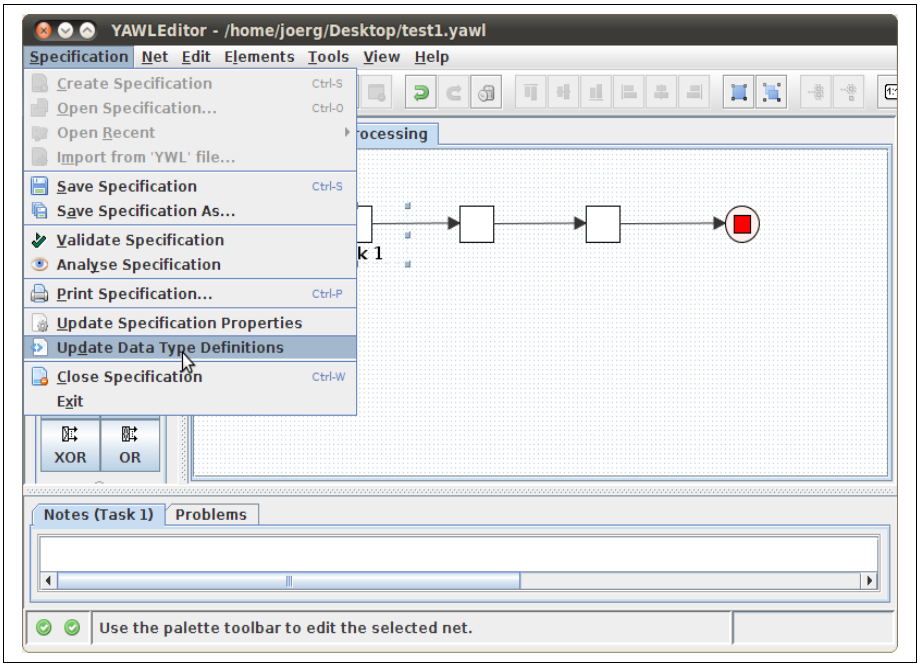
Finally, you need load the specification into the YAWL engine, just like you have done in the previous class, and execute it. YAWL will again analyze the specification for errors when it is uploaded to the engine. If all goes well, you should now see forms for data entry and data updates when you view/edit a work item in your queue. Make sure you understand how the form and data on the web page relate to the data elements that you have defined for the different tasks.



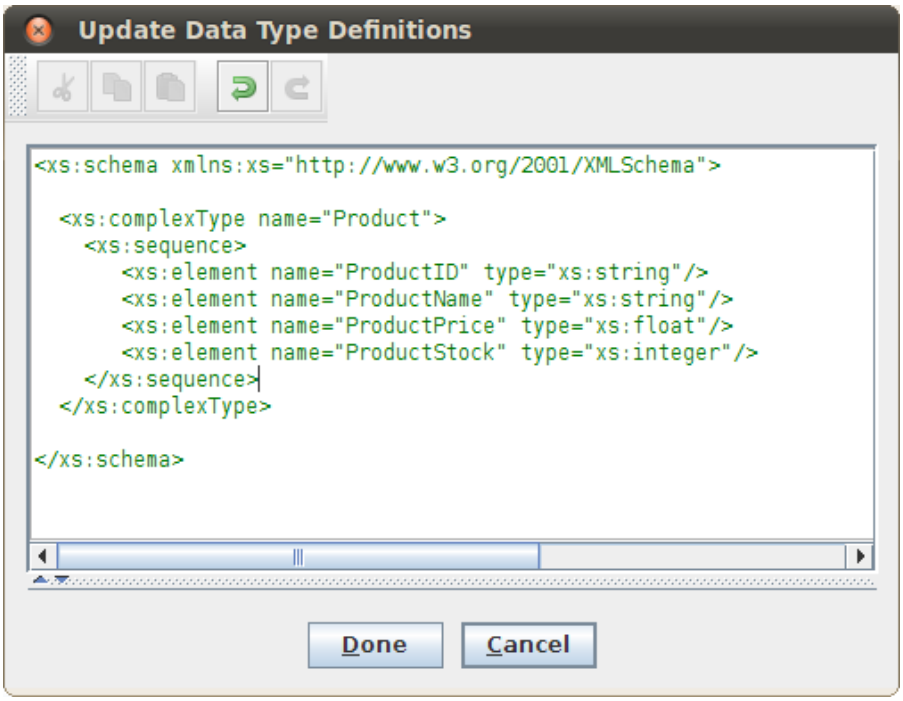
Input and output parameter mappings

Bratosin (2009)

The article by Bratosin is really a brief tutorial on creating your own data types and using them in YAWL. You should also see Section 5 of the YAWL user manual for this. I think the Bratosin article is sufficiently step-by-step that I will not repeat it here. You are encouraged (and there are some review exercises for that below) to define your own data types for your processes. In order to “install” your own data types, choose “Update data type definitions” from the “File” menu of the editor and thenwrite (or copy/paste) your definitions into the dialog box. If the definition is correct, the text in the dialog box will appear green, otherwise red.



Selecting data type definitions in the menu



Adding a complex data type to YAWL

As Bratosin points out in her paper, there are many very good tutorials on this available on the web. However, we really need only the basics of it, if that.

Review Questions

- Explain the four types of data patterns and give at least two examples for each type
- Explain what XML is
- Explain what XQuery is and the role it plays in YAWL
- What is the difference between the accessor query and the splitter query?
- What is the difference between the instance query and the aggregate query?

Review Exercises

- Chapter 8, Exercises 1, 2, 3 (focus on the data perspective)

You may have done these exercises before without the data perspective, you can now do them fully.

- Chapter 8, Exercise 5

You may have done this exercise before. You should now be able to define the necessary net and task data variables. Do this first using simple data types (the ones that are pre-defined). If you feel ambitious, then try to define your own data type for a Credit card application and use that. The data you need are at least the following:

- Customer name
- Credit limit
- Credit history (assume it is just some text)
- Customer income

As part of the process, you may also need to keep track of verification and approval status

Upload the specification and execute the process, again possibly using your group members, to verify that all data can be entered/updated as necessary.

- Chapter 2, Exercise 2

You may have done this exercise before. You should now be able to define the necessary data for a travel reservation. Do this first using simple data types. Then, define your own data type for the Travel arrangement. The data you need is at least the following:

- Traveller name
- Hotel reservations (including date in, date out, hotel name, hotel location, price)
- Car reservation (including type of car, location, pickup date, return date, price)
- Flight reservation (including flight number, date, price)
- Payment information (including date paid)

You may find other relevant information in addition to these, and you may want to have multiple of these, i.e. two or more hotel reservations or flights, as part of the same travel arrangement.

Upload the specification and execute the process, again possibly using your group members, to verify that all data can be entered/updated as necessary.

- Chapter 2, Exercise 3

You may have done this exercise before. You should now be able to define the necessary data for an improvement recommendation. Because the exercise is relatively simple, you may not need much data, but you should still try and define your own datatypes.

Upload the specification and execute the process, again possibly using your group members, to verify that all data can be entered/updated as necessary.

- Chapter 2, Exercise 4

You may have done this exercise before. You should now be able to define the necessary data for an accident investigation. You can do this first using simple data types. Then, define your own data type for the Accident Investigation. The data you need is at least the following:

- Date of accident
- Machinery involved (perhaps multiple?)
- Witness reports (names, reports, perhaps multiple witnesses?)
- Investigation Findings

You may find other relevant information in addition to these, and you may want to have multiples of some of these, like witness reports.

Upload the specification and execute the process, again possibly using your group members, to verify that all data can be entered/updated as necessary.