Editor 3.0: Redesigning the YAWL User Interface

Michael Adams

Queensland University of Technology, Brisbane, Australia.
mj.adams@qut.edu.au

Abstract. It has long been a concern that the wider uptake of the YAWL environment may have been hindered by the usability issues identified in the current Process Editor. As a consequence, it was decided that the Editor be completely rewritten to address those usability limitations. The result has been the implementation of a new YAWL Process Editor architecture that creates a clear separation between the User Interface component layer and the core processing *back end*, facilitating the redesign of the default user interface. This new architecture also supports the development of multiple User Interface *front ends* for specific contexts that take advantage of the core capabilities the new Editor architecture has to offer.

Keywords: YAWL, Editor, Workflow, User Interface

1 Introduction

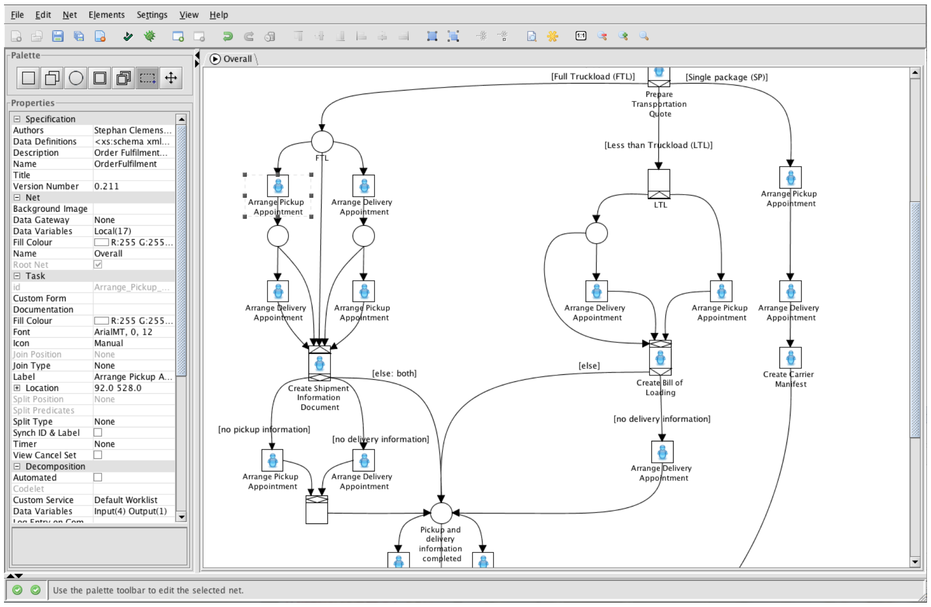
For any software, user acceptance must be of primary concern, and for most users the design and accessibility of the user interface (UI) has a much greater bearing on uptake and continued use than any functionality the software provides. For existing products, redesign of a user interface can lead to substantial improvements in learning time, performance speed, error rates and overall user satisfaction [1].

The first version of the YAWL Editor appeared almost a decade ago as a prototype, with the objective of demonstrating that it was possible to graphically design a process model in the YAWL language and have it transformed into a format that could be interpreted and executed by the YAWL Engine. Originally supporting only the control-flow perspective, many versions have since been released to include support for the data perspective and later the resource perspective, then timers, configuration, extended attributes and many other features. In each case, new functionality was 'grafted' onto the core prototypical architecture. Over time the result was a code base that had become difficult to maintain and extend, and a user interface that revealed both its age and its usability limitations.

To address these shortcomings, the YAWL Editor has been completely redesigned and is being rewritten from the ground up. The overriding driver of the project has been to make the Editor easier to use, and therefore more accessible to a wider audience. This paper outlines the main features of the new Editor.

2 Code Redesign

The original Editor architecture has a flattened package structure that mixes the user interface components with the underlying data manipulation and transportation. This meant that the user interface components were very tightly coupled to their data, which made it difficult to introduce changes to either aspect without disturbing the other as a side-effect. In addition, on opening a specification file, the Editor would first use Engine classes to parse the XML description contained in the file, then use those objects to populate its own 'mirror' classes (the Engine classes contain the code to (un)marshal to and from XML). When saving a model to file, that process was reversed. This policy introduced a large degree of redundant and error prone code, an artefact of its prototypical heritage.



File
Edit
Net
Elements
Settings
View
Help
Palette
Properties
Overall
[Full Truckload (FTL)]
[Single package (SP)]
Prepare Transportation Quote
[Less than Truckload (LTL)]
FTL
LTL
Arrange Pickup Appointment
Arrange Delivery Appointment
Arrange Delivery Appointment
Arrange Pickup Appointment
Arrange Pickup Appointment
Arrange Delivery Appointment
Arrange Delivery Appointment
Arrange Pickup Appointment
Arrange Delivery Appointment
Create Shipment Information Document
[else: both]
[else]
Create Bill of Loading
Create Carrier Manifest
[no pickup information]
[no delivery information]
[no delivery information]
Arrange Pickup Appointment
Arrange Delivery Appointment
Arrange Delivery Appointment
Pickup and delivery information completed

⊟ Specification	
Authors	Stephan Clemens...
Data Definitions	<xs:schema xml...
Description	Order Fulfilment...
Name	OrderFulfilment
Title	
Version Number	0.211
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	Local(17)
Fill Colour	☐ R:255 G:255...
Name	Overall
Root Net	✓
⊟ Task	
id	Arrange_Pickup_...
Custom Form	
Documentation	
Fill Colour	☐ R:255 G:255...
Font	ArialMT, 0, 12
Icon	Manual
Join Position	None
Join Type	None
Label	Arrange Pickup A...
⊞ Location	92.0 528.0
Split Position	None
Split Predicates	
Split Type	None
Synch ID & Label	☐
Timer	None
View Cancel Set	☐
⊟ Decomposition	
Automated	☐
Codelet	
Custom Service	Default Worklist
Data Variables	Input(4) Output(1)

Use the palette toolbar to edit the selected net.

Fig. 1. An Overview of the New Editor User Interface

In the redesigned Editor there is complete separation of the UI from the underlying framework. The main advantages are twofold:

1. The completely separated core framework can now be deployed in its own jar file. This allows external developers to create new and multiple user interfaces that interact directly with the core framework through a concise API, and frees them from concerns regarding how to store and manipulate specification control flow, data and resourcing descriptors, and file handling techniques.

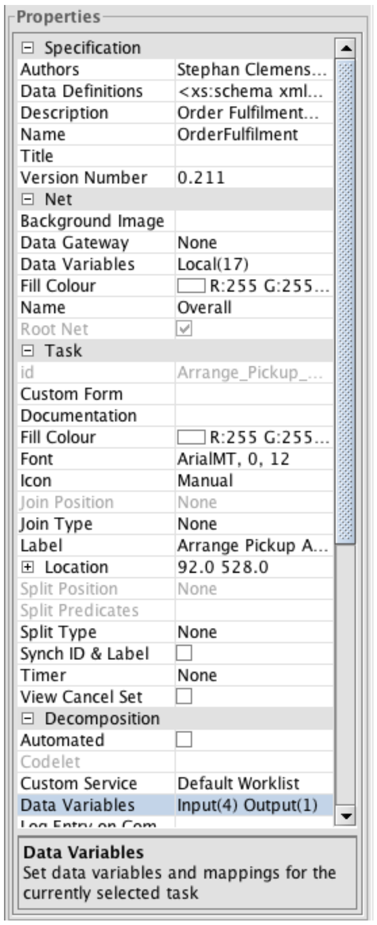
2. The core framework interacts with the YAWL Engine classes only. The negates that need for transformations when opening and saving specification files, which results in much faster file reading and writing times, and removes the scope for introducing errors into the ‘backend’ code when developing user interfaces.

3 Interface Redesign

A large factor in the usability of a software product, and particularly an graphical editor, is the ease with which the properties of an object can be set, updated and maintained. The original Editor used a number of different methods to update properties, found in various places within the interface. For example, specification properties were set via a dialog accessed from the *File* menu, and net properties from the *Net* menu, while task properties were generally accessed via a right-clicked context menu on the task itself. Adding a split or join to a task was accomplished via a box under the palette. Selecting an icon for the task was done in a different location again. Having various locations for these similar capabilities necessarily reduces usability and control.

The new Editor interface places all of the properties, for all aspects of the active process model, within a single expandable *Properties* window to the left of the graphical canvas (Figure 1). The relevant set of properties and their values for the currently selected object or objects are displayed in their appropriate form. Such properties windows are common in today’s editors, and knowing that all properties can be set in one location greatly aids ease of use.

The detailed view of the Properties window in Figure 2 shows that the currently selected object is a task with a decomposition, which therefore allows all properties of the Specification, Net, Task and Decomposition to be viewed simultaneously and easily selected for editing. Values are displayed in a format dependent on their data type. For example, simple values (e.g. Name, Version, Label) are shown as is, boolean types (e.g. Root Net, Sync, Automated)


Properties

⊟ Specification	
Authors	Stephan Clemens...
Data Definitions	<xs:schema xml...
Description	Order Fulfilment...
Name	OrderFulfilment
Title	
Version Number	0.211
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	Local(17)
Fill Colour	☐ R:255 G:255...
Name	Overall
Root Net	✓
⊟ Task	
id	Arrange_Pickup_...
Custom Form	
Documentation	
Fill Colour	☐ R:255 G:255...
Font	ArialMT, 0, 12
Icon	Manual
Join Position	None
Join Type	None
Label	Arrange Pickup A...
⊞ Location	92.0 528.0
Split Position	None
Split Predicates	
Split Type	None
Synch ID & Label	☐
Timer	None
View Cancel Set	☐
⊟ Decomposition	
Automated	☐
Codelet	
Custom Service	Default Worklist
Data Variables	Input(4) Output(1)
Log Entry on Com	

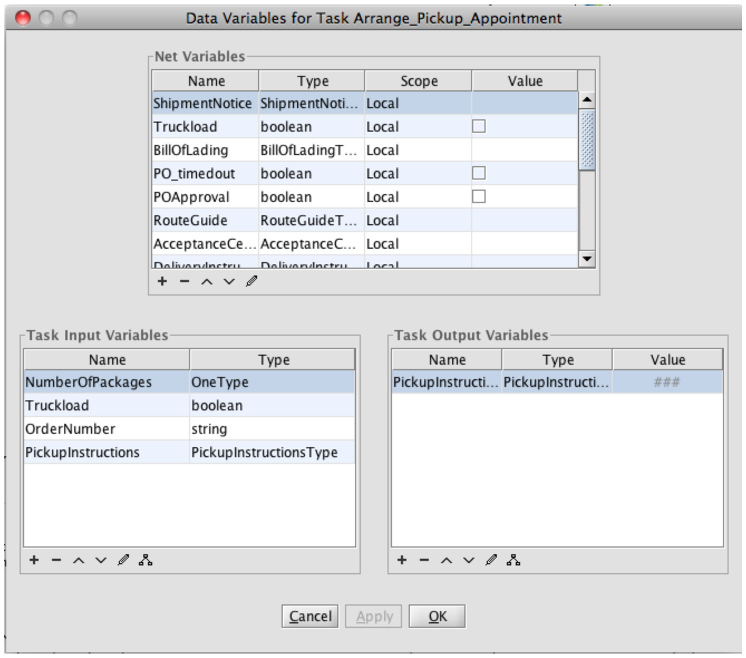
Data Variables
Set data variables and mappings for the currently selected task

Fig. 2. The Properties Window (detail)

as check boxes. More complex values, such as Fill Colour, Font and Data Variables, are shown in a summary form appropriate to their type.

Values for properties such as Data Gateway, Icon, Split and Join Types and Positions, and Decomposition are selectable via a dropdown list. Any property modification that represents a visual change is instantly updated on the canvas, and vice versa. For example, the current location coordinates for the selected task can be changed manually in the Properties window, which will immediately move the selected task to its new location on the canvas. A description of the currently selected property is displayed at the bottom of the window.

Data Perspective Arguably the most significant impediment to usability for the casual user in the original Editor was the requirement of a knowledge of XQuery for all but the most trivial of process designs. Typically, net-level variables were added via the *Net* menu, task-level variables via the *Decomposition Update* dialog (accessed from the right-click context menu) and then mapping input and outputs with XQuery parameters was achieved via the *Update Parameter Mappings* dialog (again form the context menu). Thus, a greater learning curve was needed, with each step providing more scope for error.



Net Variables

Name	Type	Scope	Value
ShipmentNotice	ShipmentNoti...	Local	
Truckload	boolean	Local	☐
BillOfLading	BillOfLadingT...	Local	
PO_timedout	boolean	Local	☐
POApproval	boolean	Local	☐
RouteGuide	RouteGuideT...	Local	
AcceptanceCe...	AcceptanceC...	Local	
DeliveryInstru	DeliveryInstru	Local	

Task Input Variables

Name	Type
NumberOfPackages	OneType
Truckload	boolean
OrderNumber	string
PickupInstructions	PickupInstructionsType

Task Output Variables

Name	Type	Value
PickupInstructi...	PickupInstructi...	###

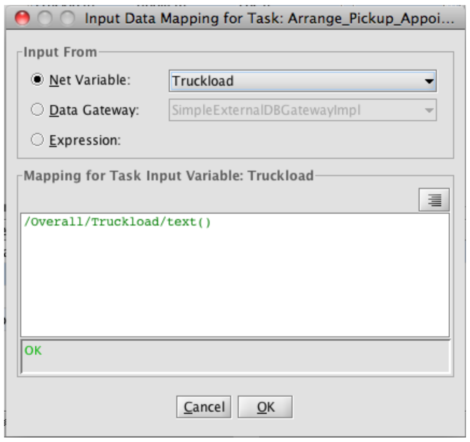
Fig. 3. The New Data Variables Dialog

In the new Editor, great care was taken to simplify this process as much as possible, through the introduction of a consolidated *Data Variables* dialog

(Figure 3). This dialog, invoked from the Task section of the Properties window, provides for the definition of net-level and task-level input and output variables all *within the same simplified dialog*. Each of the three areas in the dialog provides columns for name, type, scope and initial or default value, as appropriate, and each contains it own mini-toolbar, for adding, removing, editing and re-ordering variables. The confusing notion of *Input&Output* scoped variables has been dropped; it is now clear that a task may have a set of inputs and a set of outputs, aligning the Editor's data perspective with that of the Engine.

Notably, a net-level variable can be mapped to a task variable *simply by drag and drop*, and the required XQuery mappings are automatically created and applied. If a mapping other than the default is needed, it is able to be edited at any time with the XQuery dialog (Figure 4), invoked by the mapping button on the mini-toolbar. This dialog has also been completely redesigned to remove the unnecessary complexity of the original Editor's dialog.

An even more complicated data design exercise in the original Editor was in the definition of XQuery parameters for multiple-instance tasks, which served to inhibit the use of multiple-instance tasks for all but the most determined designers. In the new Editor, the XQuery parameters for multiple-instance tasks are also automatically generated and applied via drag and drop in the same dialog as single-instance tasks, making the capabilities offered by multiple-instance tasks available to expert and novice user alike.


Input Data Mapping for Task: Arrange_Pickup_Appoi...

Input From

(●) Net Variable: Truckload

( ) Data Gateway: SimpleExternalDBGatewayImpl

( ) Expression:

Mapping for Task Input Variable: Truckload

```
/Overall/Truckload/text()
```

OK

Fig. 4. The XQuery Mapping Dialog

The definition of complex data types using XML Schema can also prove problematic for novice designers. The new Editor will introduce a new, simplified data definition language, loosely based on *Pascal* syntax, allowing for more easily expressed data type definitions that will automatically be transformed to/from XML Schema, thus providing two discrete, user-switchable views of the same type definitions (Listings 1.1 & 1.2).

Resource Perspective The 'Wizard' metaphor used by the original Editor to capture resourcing requirements has proven less than effective in terms of user interface design. In addition to being dated in terms of UI constructs, the metaphor as deployed suffered from three main issues:

1. It was necessary to always begin at the first page and work your way through, even if you wanted only to make a small change on the final page.
2. There was no facility to undo or cancel a change.

3. There is too much information displayed on each screen. While potentially useful for novice users, the information becomes unwanted 'noise' as experience is gained. A better approach is to have the information available separately through a 'Help' dialog that can be invoked via a button.

Listing 1.1. Type Definition Expressed as XML Schema (example)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="BookOrder">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="price" type="xs:double"/>
      <xs:element name="inStock" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="BookList">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="order" type="BookOrder"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Listing 1.2. The Same Type Definition expressed in Simplified DTD Syntax

```
type BookOrder = record {
    title: string;
    price: double;
    inStock: boolean;
}
type BookList = order(1..n) of BookOrder;
```

All three concerns have been addressed in the new Editor through the use of a tabbed dialog that shows the various resourcing properties, any of which can be navigated to directly, with verbose help information removed to separately invoked dialogs. Also, the new Editor will support the *swim-lane* metaphor, allowing resources to be assigned to tasks graphically via their placement in the appropriate lanes, which will aid in the readability of models by all stakeholders.

4 Conclusion

This paper briefly outlines a few of the main features of the new YAWL Editor, however almost every aspect has been updated and improved. It is hoped that the reward for the redesign will be realised through increased deployment of the YAWL environment, and a positive affect on the teachability of YAWL, since novice users will no longer face such a steep learning curve in the use of the Editor before positive results can be achieved.

An alpha version is currently scheduled for limited release in August 2013.

References

1. B. Shneiderman, C. Plaisant, M. Cohen, and S. Jacobs. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 5th edition, 2009.