UNIVERSITE CATHOLIQUE DE LOUVAIN

FACULTE DES SCIENCES ECONOMIQUES, SOCIALES et POLITIQUES

INSTITUT D'ADMINISTRATION ET DE GESTION

LOUVAIN
School of Management

Development of a user interface generator for a workflow information system

Supervisor : Prof. Jean Vanderdonckt
Unité de Systèmes d'Information

Mémoire présenté en vue
de l'obtention du grade
de Maître
en Informatique
par : Miguel Moreno Roldán

Louvain-la-Neuve
Année académique 2008-2009

Acknowledgement

I would like to extend my thanks to Jean Vanderdocnkt, my supervisor, for his support, direction and invaluable advice.

For accepting to mentor my thesis, I would like to thank Josefina Guerrero and Christophe Lemaigre as well as Juan Manuel González and Francisco Javier Martínez, for their continuing support.

Furthermore, I would like to mention Defimedia company (of the IMmedia group) with a special note of appreciation to Luc, Benoit, Renaud, Grégory and Olivier as without their support and patience, this thesis would not have been possible.

To all those who I have met during my stay in Belgium, in particular those who are close, I am very grateful for your encouragement.

My Mum, Dad and sister whose loving support gave me courage and strength.

To all I say thank you.

Table of contents

Chapter 1 Introduction 9

1.1 Thesis context 10
1.1.1 Workflow 10
1.1.2 User interfaces 11

1.2 Objectives 13

1.3 Reading map 13

Chapter 2 State of the art 16

2.1 Workflow 17
2.1.1 Workflow models 17
2.1.1.1 Activity diagrams 17
2.1.1.2 Data flow diagrams 19
2.1.2 Workflow notations 21
2.1.2.1 UML State charts 21
2.1.2.2 Petri nets 22
2.1.2.3 YAWL 24
2.1.2.4 BPMN 26
2.1.2.5 BPEL 28

2.2 Workflow tools 29
2.2.1 Commercial software 30
2.2.1.1 Staffware 30
2.2.1.2 Websphere MQ Workflow 31
2.2.1.3 The Progression Model 31
2.2.1.4 Microsoft Windows Workflow Foundation 33
2.2.1.5 Flexo business 34
2.2.1.6 Business Process Visual Architect 36
2.2.2 IMmedia S.A. Atoms 36
2.2.3 FlowiXML Workflow editor 37

Chapter 3 eXtensible Mark-up Languages for User Interface Definitions 40

3.1 Introduction to XML 41

3.2 User Interface Description Languages based on XML 42
3.2.1 XIML 42
3.2.2 UIML 42
3.2.3 XUL 43
3.2.4 AUIML 43
3.2.5 UsiXML 43
3.2.6 User Interfaces definition in IMmedia S.A. Atoms 44

3.3 XML transformations 44
3.3.1 XSLT Stylesheet Language 44
3.3.2 Transforming XML with Java 45
3.3.2.1 SAX 46
3.3.2.2 DOM 46
3.3.2.3 JAXP 47
3.3.3 Composition of our solution 47

Chapter 4 Development of a user interface generator for a workflow information system 48

4.1 Overview of the global approach 49

4.2 Workflow resource patterns 50
4.2.1 Creation patterns 51

4.2.1.1 Direct allocation 52
4.2.1.2 Deferred allocation 52
4.2.1.3 Authorization based 53
4.2.1.4 Separation of duties 54
4.2.1.5 Case handling 54
4.2.1.6 Retain familiar 55
4.2.1.7 Capability-based allocation 56
4.2.1.8 History-based allocation 56
4.2.1.9 Hierarchy level based 57
4.2.2 Push patterns 57
4.2.2.1 Distribution by offer single-resource 58
4.2.2.2 Distribution by offer multiple-resources 58
4.2.2.3 Distribution by allocation single-resource 59
4.2.2.4 Random allocation 59
4.2.2.5 Round robin allocation 60
4.2.2.6 Shortest queue 61
4.2.2.7 Early distribution 61
4.2.2.8 Distribution on enablement 62
4.2.2.9 Late distribution 62
4.2.3 Pull patterns 63
4.2.3.1 Resource-initiated allocation 63
4.2.3.2 Resource-initiated execution-allocated task 64
4.2.3.3 Resource-initiated execution-offered task 64
4.2.3.4 System determined agenda content 65
4.2.3.5 Resource determined agenda content 65
4.2.3.6 Selection autonomy 66
4.2.4 Detour patterns 66
4.2.4.1 Delegation 66
4.2.4.2 Escalation 67
4.2.4.3 Deallocation 68
4.2.4.4 Stateful reallocation 68
4.2.4.5 Stateless reallocation 69
4.2.4.6 Suspension / resumption 70
4.2.4.7 Skip 70
4.2.4.8 Redo 71
4.2.4.9 Pre-Do 72
4.2.5 Auto-start patterns 72
4.2.5.1 Commencement on creation 73
4.2.5.2 Commencement on allocation 73
4.2.5.3 Piled execution 74
4.2.5.4 Chained execution 74
4.3 XSL transformations from Petri nets to State charts 75
4.3.1 Output from FlowiXML workflow editor 75
4.3.1.1 processModel 75
4.3.1.2 taskModel 76
4.3.1.3 mappingModel 77
4.3.1.4 workflowModel 78
4.3.2 XML workflow definition of IMmedia Atoms 79
4.3.3 XSL transformation 80
4.3.3.1 Petri nets to state charts 80
4.3.3.2 Patterns representation 82
4.3.3.3 User interfaces 84
Chapter 5 Case studies 86
5.1 Case study 1 - Simplified buying request 87
5.2 Case study 2 - Complete buying request 90
Chapter 6 Conclusion 93

6.1 Contribution **94**
6.2 Future work **94**
References ***95***
Annex. Attached content ***98***

Table of figures

Figure 1-1 An example of a workflow process ____ 10
Figure 1-2 User interface of MS Office 2007, easy to understand and to use ____ 11
Figure 1-3 An example of a bad user interface, maybe designed thinking in a hardware expert, but to provide software like this to the end users is not good ____ 12
Figure 1-4 Graphical representation of the main goal of this thesis ____ 13
Figure 1-5 Graphical representation of the reading map of this thesis, divided by different levels of experience of the Readers ____ 15
Figure 2-1 Activity diagram example ____ 18
Figure 2-2 Example of how to use the Yourdon & Coad notation for data flow diagrams ____ 20
Figure 2-3 The nesting of data flow layers in Yourdon & Coad notation ____ 20
Figure 2-4 An example of UML State charts diagram ____ 22
Figure 2-5 Petri net modelling the simplified workflow of a waiter in a café ____ 23
Figure 2-6 A Segment of a Process with Data Objects, Groups, and Annotations ____ 28
Figure 2-7 Example BPEL process for travel arrangements ____ 29
Figure 2-8 Graphical Workflow Definer of Staffware ____ 30
Figure 2-9 Buildtime user interface ____ 31
Figure 2-10 The progression analyzer displaying the Enter Name scene ____ 33
Figure 2-12 The Visual Studio 2005 Workflow designer ____ 34
Figure 2-13 An interior view of the OrderCreatedEvent EventDriven activity ____ 34
Figure 2-14 The workflow modeller ____ 35
Figure 2-15 Layout of user graphical interfaces editor ____ 35
Figure 2-16 User interface of BP-VA ____ 36
Figure 2-17 A screenshot of the Atoms workflow editor ____ 37
Figure 2-18 A screenshot of the FlowiXML workflow editor ____ 38
Figure 2-19 The main user interface ____ 38
Figure 2-20 Workflow editor user interface ____ 39
Figure 3-1 An example of a bookstore definition in a XML file ____ 41
Figure 3-2 An example of XSLT stylesheet to transform the last XML file definition (Figure 3-1) in a HTML webpage ____ 45
Figure 3-3 The output obtained after apply our XSLT stylesheet (Figure 3-2) to the last XML file definition (Figure 3-1) ____ 45
Figure 4-1 Task represented in FlowiXML workflow editor ____ 49
Figure 4-2 Direct allocation workflow resource pattern represented in terms of petri nets ____ 49
Figure 4-3 Petri nets process transformed into state charts in Atoms ____ 49
Figure 4-4 Creation patterns ____ 52
Figure 4-5 Push patterns ____ 58
Figure 4-6 Pull patterns ____ 63
Figure 4-7 Detour patterns ____ 66
Figure 4-8 Auto-start patterns ____ 72
Figure 4-9 Task with a direct allocation pattern applied ____ 81
Figure 4-10 Representation of a task with direct allocation pattern applied ____ 81
Figure 4-11 Possible solution to AND-split problem ____ 82
Figure 4-12 Conflictive process transformation when an AND-split is found ____ 82
Figure 4-13 Reallocation patterns definition ____ 83
Figure 4-14 Skip pattern definition ____ 83
Figure 4-15 Redo pattern definition ____ 83
Figure 4-16 Direct allocation pattern definition ____ 84
Figure 4-17 User interface to select a resource ____ 84
Figure 4-18 Reallocation patterns transformation ____ 84
Figure 4-19 Skip pattern transformation ____ 85
Figure 4-20 Redo pattern transformation ____ 85
Figure 4-21 Direct allocation patterns transformation ____ 85
Figure 5-1 Process of a simplified buying request in an online shop ____ 87
Figure 5-2 Simplified buying request process transformed to state charts in Atoms ____ 88
Figure 5-3 Use interface of the select resource pattern task ____ 89
Figure 5-4 Process of a complete buying request in a company ____ 91
Figure 5-5 Complete buying request process transformed in state charts in Atoms ____ 92

Table of tables

Table 2-1 Activity diagrams syntax 18
Table 2-2 Data flow diagrams syntax 19
Table 2-3 UML State charts notation 21
Table 2-4 Petri nets notation 23
Table 2-5 YAWL notation 26
Table 2-6 BPMN syntax notation 28
Table 4-1 Template to define workflow patterns 50
Table 4-2 Summary of the workflow resource patterns analysis 51
Table 4-3 Direct allocation analysis 52
Table 4-4 Deferred allocation analysis 53
Table 4-5 Authorization pattern analysis 53
Table 4-6 Separation of duties pattern analysis 54
Table 4-7 Case handling pattern analysis 55
Table 4-8 Retain familiar pattern analysis 55
Table 4-9 Capability-based allocation pattern analysis 56
Table 4-10 History-based allocation pattern analysis 57
Table 4-11 Hierarchy level based pattern analysis 57
Table 4-12 Distribution by offer single resource pattern analysis 58
Table 4-13 Distribution by offer multiple resource pattern analysis 59
Table 4-14 Distribution by allocation single resource pattern analysis 59
Table 4-15 Random allocation pattern analysis 60
Table 4-16 Round robin allocation pattern analysis 60
Table 4-17 Shortest queue pattern analysis 61
Table 4-18 Early distribution pattern analysis 61
Table 4-19 Distribution on enablement pattern analysis 62
Table 4-20 Late distribution pattern analysis 62
Table 4-21 Resource-initiated allocation pattern analysis 63
Table 4-22 Resource-initiated execution-allocated task pattern analysis 64
Table 4-23 Resource-initiated execution-offered task pattern analysis 64
Table 4-24 System determined agenda content pattern analysis 65
Table 4-25 Resource determined agenda content pattern analysis 65
Table 4-26 Selection autonomy pattern analysis 66
Table 4-27 Delegation pattern analysis 67
Table 4-28 Escalation pattern analysis 68
Table 4-29 Deallocation pattern analysis 68
Table 4-30 Stateful reallocation pattern analysis 69
Table 4-31 Stateless reallocation pattern analysis 69
Table 4-32 Suspension/resumption pattern analysis 70
Table 4-33 Skip pattern analysis 71
Table 4-34 Redo pattern analysis 71
Table 4-35 Pre-do pattern analysis 72
Table 4-36 Commencement on creation pattern analysis 73
Table 4-37 Commencement on allocation pattern analysis 73
Table 4-38 Piled execution pattern analysis 74
Table 4-39 Chained execution pattern analysis 74
Table 4- 40 Summary of the relations between object from FlowiXML workflow editor and Atoms 81

Chapter 1 Introduction

Nowadays, everyone interacts with personal computers everyday. We use it to work, to enjoy, to look for any kind of information… The computers are no longer electrical appliances created exclusively for experts like in its beginning, everyone can become a good user. We have to give thanks to user interfaces, which everyday improves its usability and accessibility.

In the Business World, the great majority of enterprise raises a certain volume of needs, which must be satisfied with support of computer application to run all processes right and with good performance. In these enterprises, the users do not need to be computing experts.

The automatic generation of user interfaces that facilitates the design of these work processes is the goal we are going to address here.

In this chapter, we start to explain the basics concepts, in a thesis context. Afterwards, we explain the objective of the past explanation.

1.1 Thesis context

1.1.1 Workflow

In the Middle Ages, the monks were copying carefully the documents that were written in planks. The Father Superior was doing the assignments of the work, maybe giving the first page of a section to the most expert artist, if the task was required a perfect performance, maybe giving it to the first one he was finding. After, he was doing the assignments of the tasks to be corrected to the top learned.

A few changes have happened in the next centuries. In any work, the supervisors assign the work, maybe basing their decisions on the training, abilities and experiences, to the resources. At the beginning, the resources were only the human being, helped possibly by the tools like typewriters or calculators.

Nowadays, the resources are anything that can carry out a task. Any kind of machine can be a resource which can receive a request to perform a task, and an information system is the one that sends it the request. These kinds of information systems are the Workflow Management Systems. Helped by a system like this, the responsible of a concrete work process can monitor it, knowing how the process is being executed and taking decisions if he thinks there is any kind of problem, like the lack of time or an increase of the cost of the performance.

For example, we can imagine the process to follow when we buy something in an online shop. Figure 1-1 is a good generic example of how we can do that, starting visiting the online shop and adding some goods to our shopping basket. Afterwards, the online shop asks us for our user identifier and password and invites us to register if we do not have registered before. Once we have been registered at the online shop, we can checkout to place the order definitively. Until last step, the costumer have been the only resource to carry out all the tasks to do, but now, the workers of the shop's warehouse start to prepare the order. And when the order is ready to be shipped, maybe the workers of the delivery department are the responsible to send the goods to the costumer. During the process, we could monitor how many orders are waiting for be prepared or how many orders are ready to be shipped.

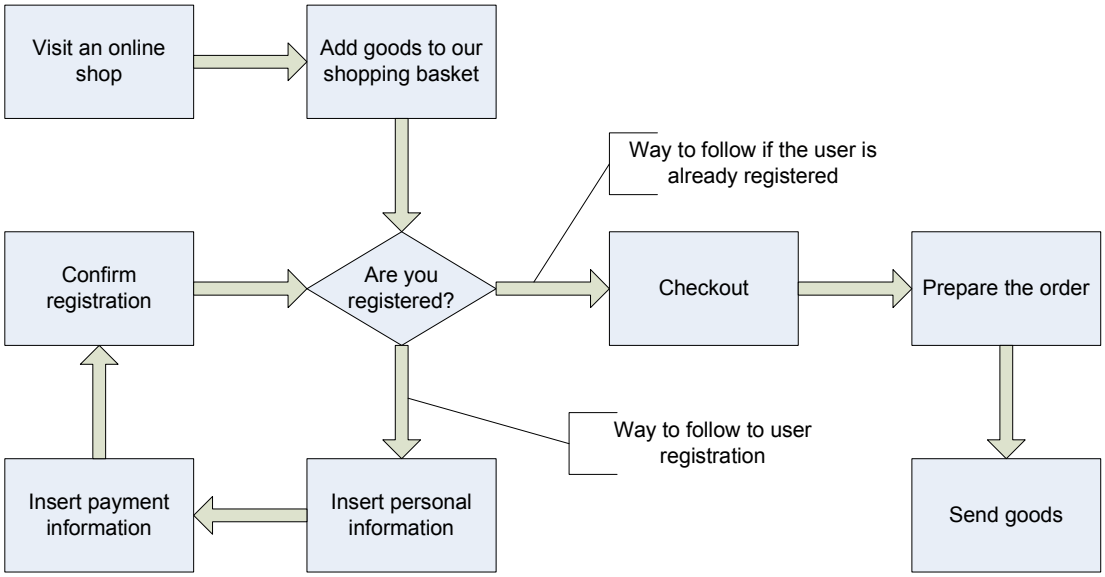


Figure 1-1 An example of a workflow process

1.1.2 User interfaces

The user interface of an application is the hardware and software elements of a computer that present the information to the user and allows him to interact with the information and the computer.

Each step (or task) in a workflow has its own user interface. The website of the shop where we were buying (Figure 1-1) is the way with which it interacts with us. Maybe, the main page of the website would show us the bestsellers products and the checkout page would show us the content of our shopping basket and the total amount including shipping costs. They would be the different user interfaces for these two different tasks.

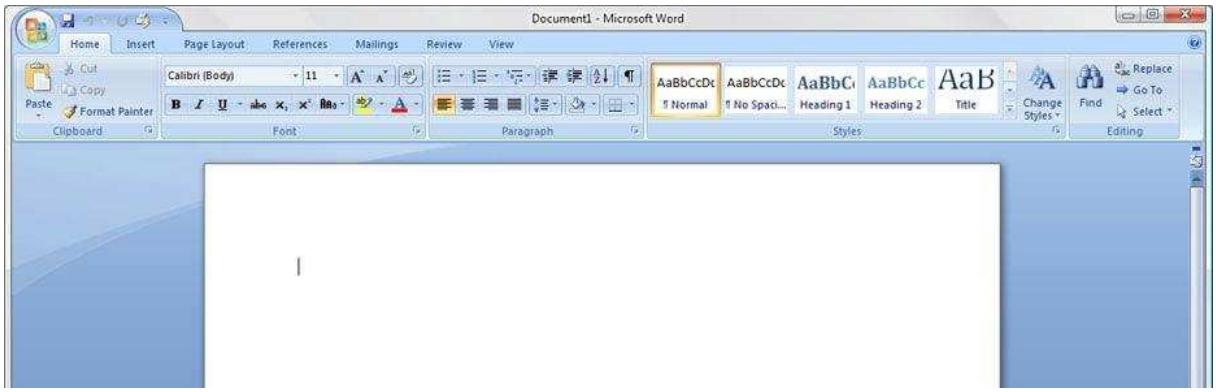


Figure 1-2 User interface of MS Office 2007, easy to understand and to use

The most popular software applications have improved, release after release, their user interfaces according with the new technologies that appear nowadays to obtain the users approval and to become more and more popular yet. Obviously, the user interfaces are not the only determining factor in the way to reach the user's approval, but it is the first one which the user will interact with, the user first impression. Figure 1-2 show us the user interface of the most popular word processing software.

An application can be very well-designed and it can generate a big set of useful results. If its user interface is well-designed too, the user will obtain a satisfactory experience after using it. In the other hand, if the user interface is not well-designed, the interaction with this application can be an unfulfilling experience because anyone will not be able to adapt oneself to this application. The useful results can not be obtained by the users. See Figure 1-3 like an example of a bad user interface.

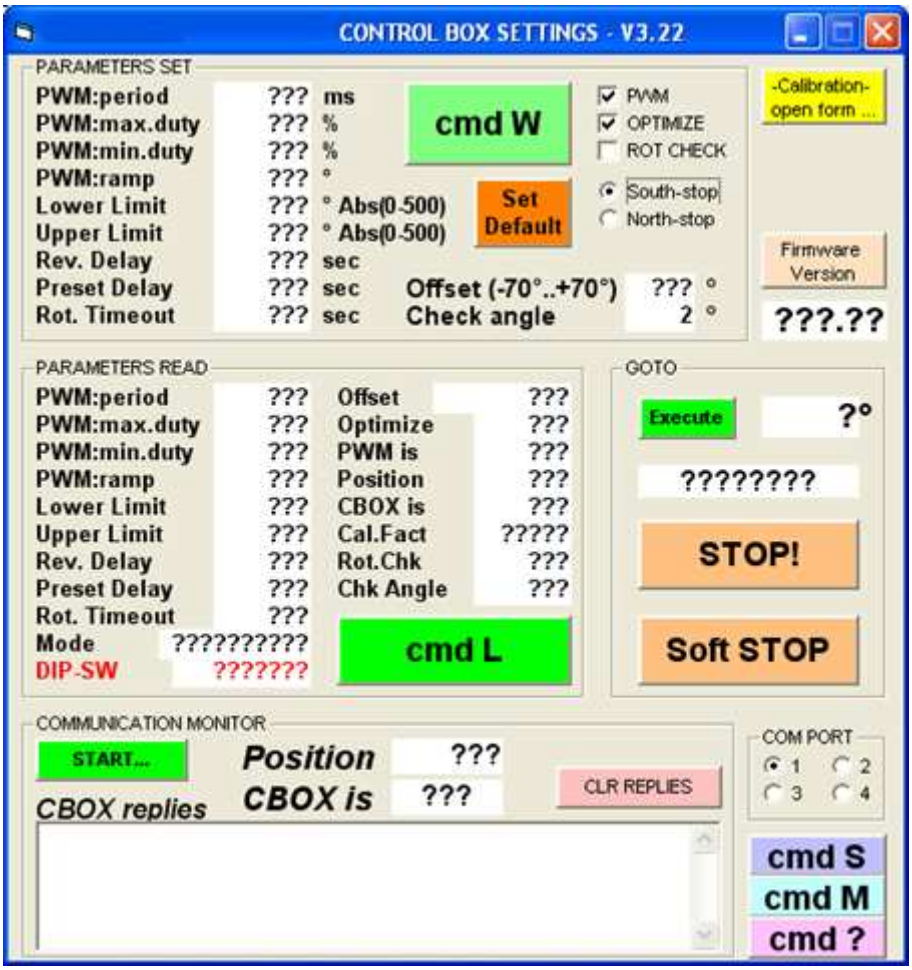


Figure 1-3 An example of a bad user interface, maybe designed thinking in a hardware expert, but to provide software like this to the end users is not good [BuiG08]

At this point, we can differentiate between two different kinds of user interfaces, in terms of design:

1. **Bad user interfaces**: The user can not understand entirely the options that the application offers. Some buttons have not a correct name related with the function it does, some menus are too much big and they contain too many options, there are too many text boxes to fill labelled with very contract names, etc (see Figure 1-3).
2. **Well-designed user interfaces**: The user interface is easy to understand and to use and the user has obtained a satisfactory experience after using it (see Figure 1-2).

Each application needs to have defined a user interface. In some applications, which need a set of data, maybe divided in some steps like in the workflow management systems, it is possible to generate the user interfaces automatically, making easier the design of the application. With automation like this, every user could define these applications without considering the user interface definition and with no programming skills.

In this thesis, we will focus only on user interfaces, which will allow us to allocate every task of the process to the available resources, in a workflow information system, automating their generation since a low-level definition of them. There are already defined the patterns that covers the whole set of cases. For each pattern, there is defined too a user interface proposition, easy to understand and to use. You can find these definitions in Chapter 4, where we show the solution we have found for each case.

1.2 Objectives

The main goal of this thesis is to generate automatically the allocation tasks user interfaces of a workflow management system, directly derivated from the workflow definition information.

Specifically, we are going to transform the data of a workflow editor which works with Petri nets notation into the data specification of another workflow editor, called Atoms from the Defimedia company, which works with State charts notation and another user interface definition. During this transformation, we will generate the necessary user interfaces to assign each task, depending on the pattern defined in the task creation.

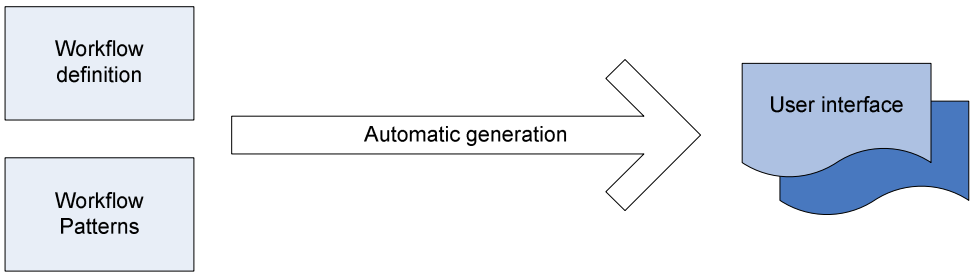


Figure 1-4 Graphical representation of the main goal of this thesis

Deriving from the main motivation of this thesis, and being necessary to reach this user interfaces automatic generation, the main steps are the followed ones:

- Familiarize me with the propositions of user interface definition in XML, especially in UsiXML.
- Familiarize me with how to represent workflow processes, especially in the FlowiXML workflow editor and in the Atoms software from IMmedia S.A.
- Study the workflow resource patterns, defined for each allocation case.
- Study the different notations to define a workflow, especially UML State charts and Petri nets.
- Study the alternatives of XML transformations and apply my knowledge in a real case, transforming a workflow defined in Petri nets into a special UML State charts one.
- Study how to generate automatically a user interface, taking the appropriate decisions, depending on different cases defined by the workflow resource patterns.

1.3 Reading map

The remainder of this thesis is structured as follows:

In chapter 1, the introduction, we introduce some necessary concepts that we will use in the next pages. The most important ones are the workflow and user interface concepts. Moreover, we define the main objectives of this thesis and then we show how it is organized.

In chapter 2, the state of the art, we introduce the highest level of knowledge about some technologies we are going to use, they have a directly relation with the ones we are going to use or simply they could be alternatives. Specifically, we will introduce the workflow models and notations, the commercial workflow management systems at present and the two ones we will use in the development, and the user interfaces for workflow management systems.

In chapter 3, XML user interface definitions, we introduce the eXtensible Mark-up Language (XML) first. Afterwards, we show what different options we can use to read and transform this language. In the end, we introduce some XML-compliant mark-up languages to describe user interfaces for multiple context of use, going into UsiXML, the one we use in the development, in depth.

In chapter 4, the development, we start with an overview of the goal of this thesis, to show after all the workflow resource patterns, explaining what user interface we have found to transform it in the target application. Afterwards, we will show the output of the source workflow editor, explaining how it stores the information, and then the input we have to generate to be read by the target application. Finally, we will show how the transformation is done.

In chapter 5, the case studies, we explain how the transformation is done in two examples, one very simple explained in full detail, and another more extended with less details.

In chapter 6, the conclusions, we will start explaining what contributions has been made by this thesis, to propose after some future work still to do which could bring this problem to new frontiers of solution.

The reader can skip some chapters or sections according on his experience. Figure 1-5 can help to decide how to read this thesis.

Figure 1-5 Graphical representation of the reading map of this thesis, divided by different levels of experience of the readers

Chapter 2 State of the art

The former chapter has already introduced the workflow, workflow management system and user interface concept. In this chapter, we are going to introduce the highest level of knowledge about some technologies we are going to use, they have a directly relation with the ones we are going to use or simply they could be alternatives.

We start with a deeply workflow introduction (deeper than in the former chapter), introducing the most important defined models and notations. After, we will explain how are the applications we are going to work with, Atoms from Defimedia company and the FlowiXML workflow editor [Lema07]. A brief introduction of commercial software is accessible at the end of the workflow tools section too.

2.1 Workflow

The Workflow Management Coalition [WfMC99] defines the workflow concept like *"the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"*.

The essential workflow characteristics are: tasks (or activities) that are performed by role-playing persons, using supporting tools that give access to a variety of shared information resources [Mars97].

Workflow technologies facilitate modelling, redesigning and administration of process in an organization [Eich04]. Due to the importance of workflow nowadays, several models have been proposed to design and specify it. In addition, workflow patterns have been identified to enrich this specification by introducing resources representation like workflow resources patterns, and for routing workflow constructions like workflow patterns [vand03].

2.1.1 Workflow models

In this section, we will briefly introduce some models that can be used to represent a workflow. There are many different types of workflow models: activity diagrams, data flow diagrams, function chaining graphs, etc.

We will detail activity diagram and data flow diagram in depth because they are two important models we need to know to understand UML state charts and Petri nets.

2.1.1.1 Activity diagrams

The UML provides two different kinds of state machine formalisms: activity diagrams and state charts. They differ in the kinds of situations to which they are applied. Activity diagrams are appropriate when the object or operation changes state primarily upon completion of the activities executed within the state rather than the asynchronous occurrence of events. We will talk about state charts in the next section (2.1.2.1 Workflow notations).

An activity diagram is used to display the sequence of activities. Activity diagrams are appropriate for showing workflows of activities and actions, from a start point to the finish point, with support for choice, iteration and concurrency, and detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities. Activity diagrams are useful for business modelling where they are used for detailing the processes involved in business activities. It shows the overall flow of control.

To design an activity diagram we have to use the next syntax definition (see Table 2-1).

Notation	Definition	Graphical representation
Initial Activity	This shows the starting point or first activity of the flow. Denoted by a solid circle.	
Activity	Represented by a rectangle with rounded or	

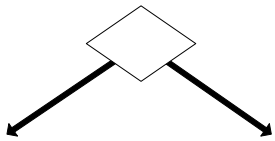
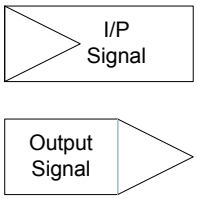
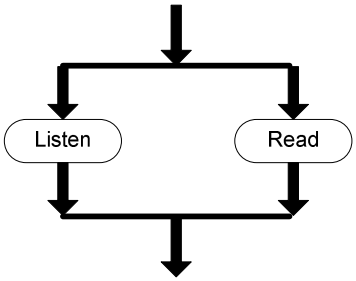
	oval edges.	
Decisions	A logic where a decision is to be made is depicted by a diamond, with the options written on either sides of the arrows emerging from the diamond, within box brackets.	
Signal	When an activity sends or receives a message, that activity is called a signal. Signals are of two types: Input signal (Message receiving activity) shown by a concave polygon and Output signal (Message sending activity) shown by a convex polygon.	
Concurrent Activities	Some activities occur simultaneously or in parallel. Such activities are called concurrent activities. This is represented by a horizontal split and the two concurrent activities next to each other, and the horizontal line again to show the end of the parallel activity.	
Final Activity	The end of the activity diagram is shown by an “eye symbol”, also called as a final activity.	

Table 2-1 Activity diagrams syntax

An example about the enrolment in a university using activity diagrams notations is defined as follows (see Figure 2-1).

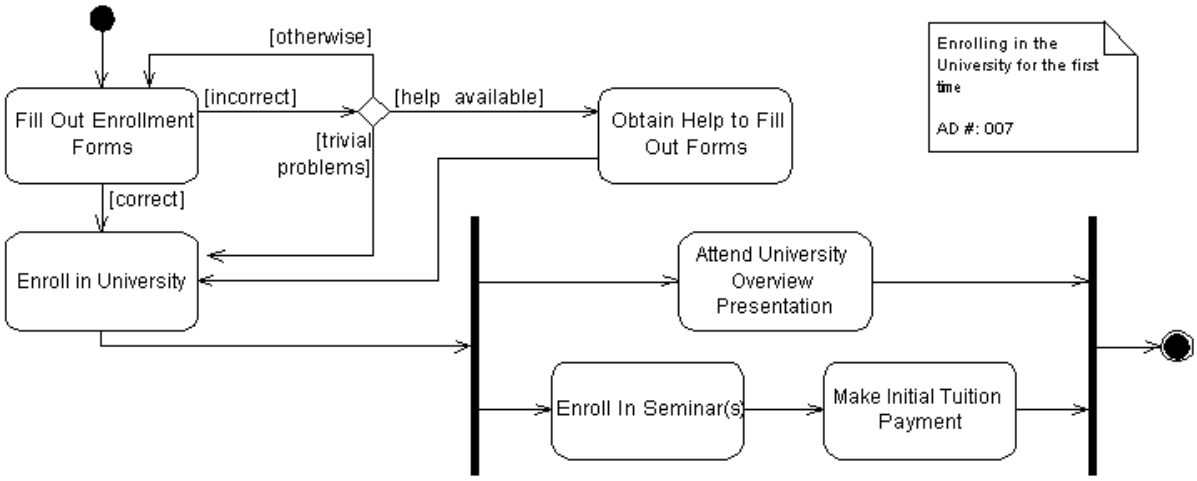


Figure 2-1 Activity diagram example [Agil08]

2.1.1.2 Data flow diagrams

Data flow diagrams were invented by Larry Constantine (born in 1943), who is considered one of the pioneers of computing and the original developer of structured design.

In the late 1970s data flow diagrams were introduced and popularized for structured analysis and design. Data flow diagrams show the flow of data from external entities into the system, showed how the data moved from one process to another, as well as its logical storage.

There are two different types of notations to define data flow diagrams: Yourdon & Coad or Gane & Sarson (see Table 2-2).

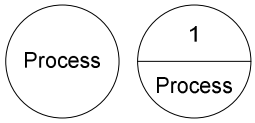
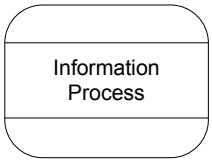
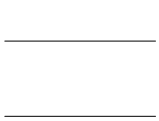
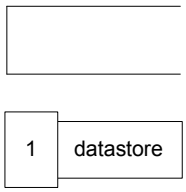
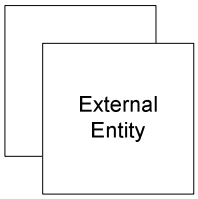
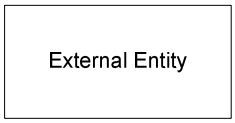
		Graphical representation	
Notation	**Description**	**Yourdon & Coad**	**Gane & Sarson**
Process	A process take data as input, do something to it, and output it.		
DataStore	Datastores are repositories of data in the system such as databases or XML files and physical stores such as filing cabinets or stacks of paper.		
Dataflow	Dataflows are pipelines through which packets of information flow. They can either be electronic data or physical items.		
External Entity	External entities are objects outside the system, with which the system communicates. External entities are sources or destinations of the system's inputs and outputs.		

Table 2-2 Data flow diagrams syntax

Now, we can see an example of how to use the Yourdon & Coad notation (see Figure 2-2).

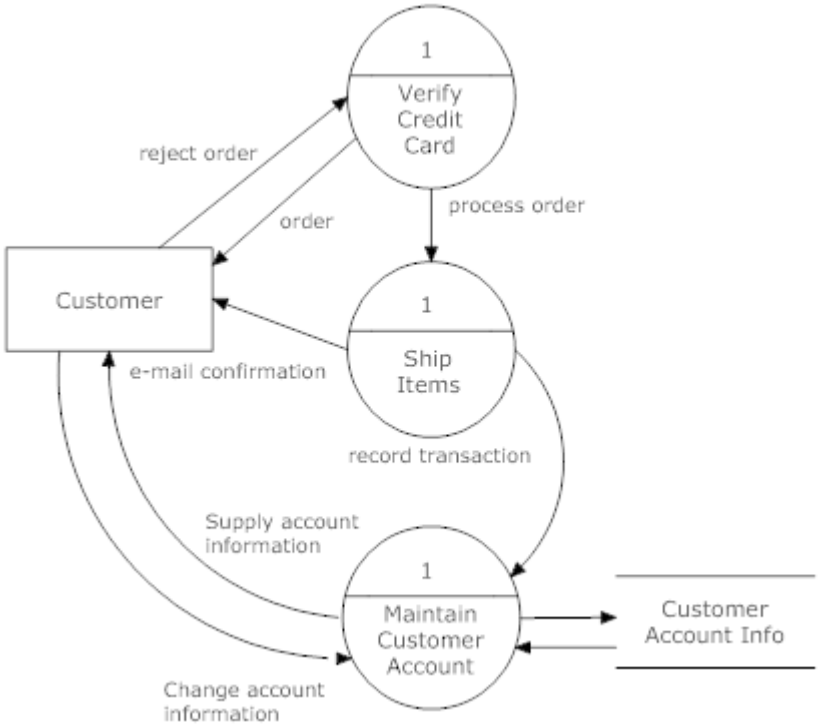


Figure 2-2 Example of how to use the Yourdon & Coad notation for data flow diagrams

A single process node on a high level diagram can be expanded to show a more detailed data flow diagram in nested layers (see Figure 2-3).

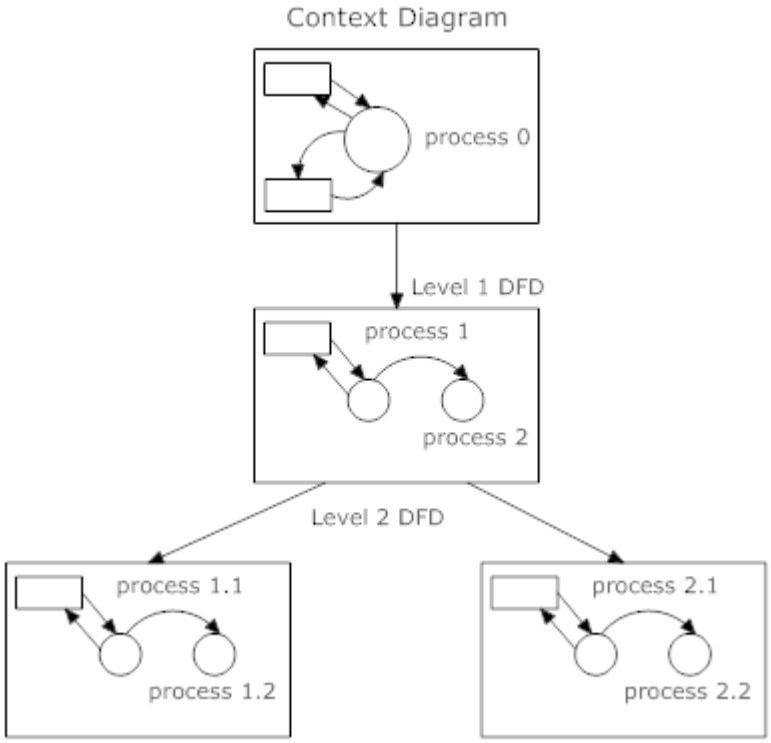


Figure 2-3 The nesting of data flow layers in Yourdon & Coad notation

2.1.2 Workflow notations

In this section, we are going to introduce the most popular workflow notations, UML State charts and Petri nets. UML State charts represent a state machine and Petri nets are a tool for modelling and analyzing processes. Both can describe processes in a graphical way.

2.1.2.1 UML State charts

In section 2.1.1.1 we have introduced a kind of state machine representation that UML provides, the Activity diagrams. In the other hand, state charts are used when the transition from state to state takes place primarily when an event of interest occurs and is more commonly used.

UML State charts were introduced by David Harel to overcome the shortcomings of prior state machine representations by adding hierarchy, concurrency and communication. State charts were adopted as a behavioural diagram within UML specification.

State charts consist of three primary things: states, transitions, and actions.

1. **States**. States are distinguishable conditions of existence that persist for a significant period of time.
2. **Transitions**. Transitions are the means by which objects change states in respond to events.
3. **Actions**. State machines also execute actions (atomic behaviours) at various points in a state machine, such as when an event triggers a transition, when a state is entered or when a state is exited.

Notation	Definition	Graphical representation
State	A state marks a mode of the entity. The graphical representation is a rectangle with rounded corners and the name of the state.	
Transition	A transition marks the changing of the object state, caused by an event. The representation is an arrow with the Event Name.	
Initial state	The initial state is the state of an object before any transitions; it is marked using a solid circle. Only one initial state is allowed on a diagram.	
Final state	The final states mark the destruction of the object whose state we are modeling. The final state is drawn using a solid circle with a surrounding circle.	

Table 2-3 UML State charts notation

Now, we can see an example of a process defined in UML State charts terms (see Figure 2-4).

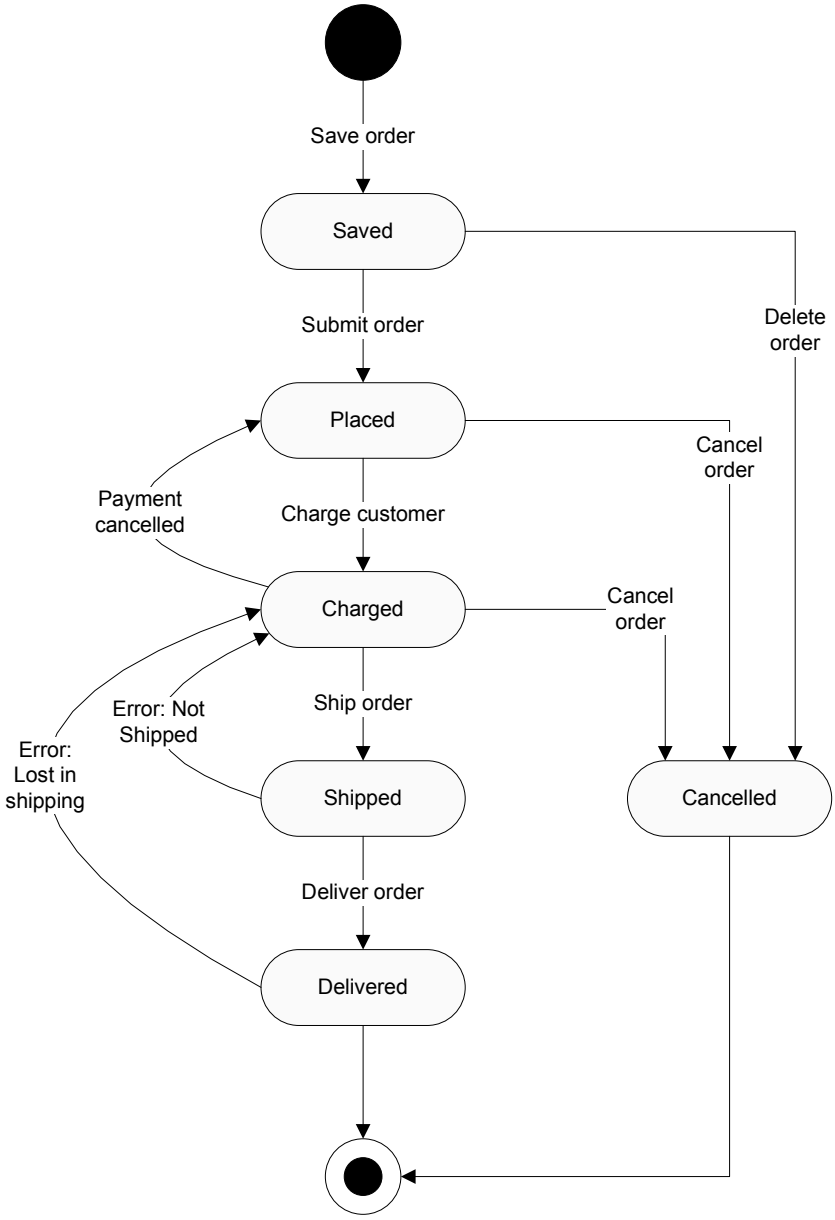


Figure 2-4 An example of UML State charts diagram

Figure 2-4 shows the different states of a current online order and its transitions. The order starts in the Saved state and arrive at the Delivered state is its goal. During this way to reach the goal, the payment could be cancelled or some errors could happen, sending back the actual state to previous states; and in some states it is possible to cancel the order, preventing the order reaching its goal.

2.1.2.2 Petri nets

Petri nets, introduced by Carl Adam Petri [Carl62], provide an elegant and useful mathematical formalism for modelling concurrent systems and their behaviours. Petri nets are graphical and mathematical modelling tools. As a graphical tool, Petri nets can be used as a visual-communication aid. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behaviour of systems [Mura89].

The graphical representation Petri net is a directed bipartite graph with two kinds of nodes: places and transitions [vand98].

1. **Places**, which are usually model resources or partial state of the system.
2. **Transitions**, which are model state transitions and synchronizations.

The nodes are connected via directed arcs and they always connect nodes of different types. The state of the system is modelled by marking the places with tokens and a place can be marked with a finite number (possibly zero) of tokens.

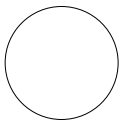
Notation	Definition	Graphical representation
Transition	Transitions are the active components of a Petri net. The triggering of a transition results in the state of the network being changed.	
Arc	Places and transitions can be linked using directed arc.	
Token	The state of a Petri net is determined by the distribution of tokens amongst the places.	
Place	Places are the passive components of a Petri net. A place may contain none, one or more tokens.	

Table 2-4 Petri nets notation

Now, we can see an example of a process defined in Petri nets terms (see Figure 2-5).

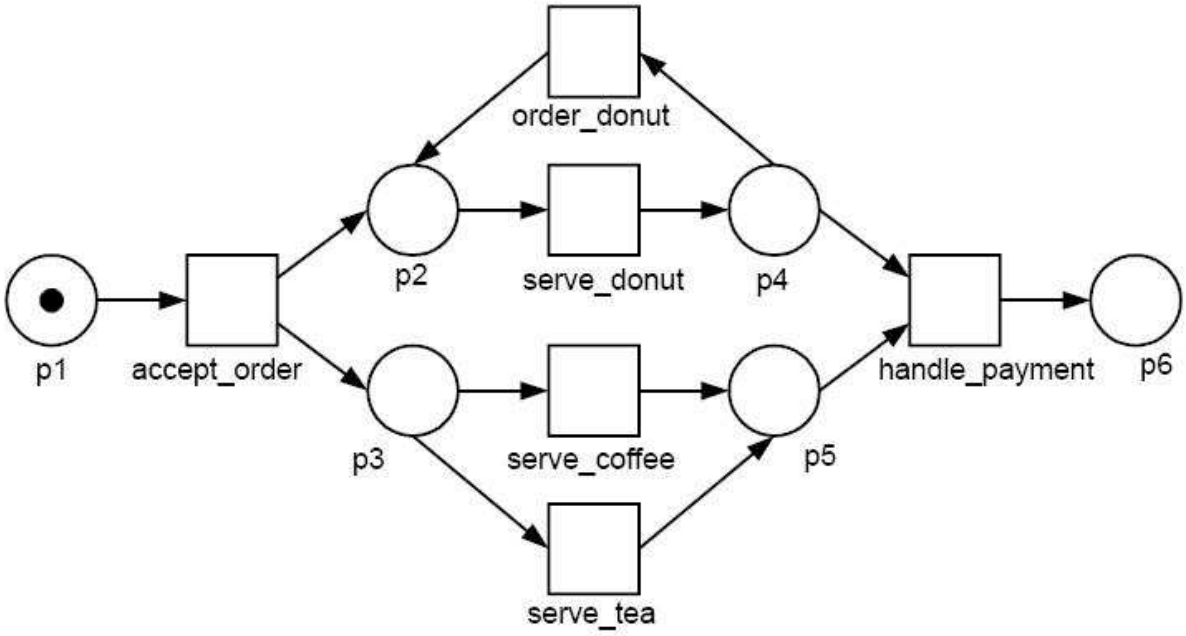


Figure 2-5 Petri net modelling the simplified workflow of a waiter in a cafe [vand01]

Figure 2-5 represents the work, but simplified, that a waiter has to do in a cafe. After accept an order, the token will wait in p2 and p3 to do the next task, serve a coffee, a tea or a donut. Then it will rest in p5, in case of serve a coffee or tea, or p4, in case of serve a donut.

Using the classical Petri net notation, it is possible to model states, events, conditions, synchronization, parallelism, choice, and iteration. In the other hand, classical Petri net does not allow the modelling of data and time. To solve these problems, some extensions have been proposed: the extension with color to model data, the extension with time, and the extension with hierarchy to structure large models are three well-known extensions of the basic Petri net model. A Petri net extended with a combination of the three extensions mentioned is called a high-level Petri net [Guer06].

1. **Extension with color**. Tokens often represent objects (cases) in the system. If we want to represent attributes that are not easily represented by a token in a classical Petri net, the net model is extended with color. In a colored Petri net each token has a value often referred to as 'color'. Transitions determine the values of the produced tokens on the basis of the values of the consumed tokens. For example, a transition describes the relation between the values of the input tokens and the values of the output tokens.
2. **Extension with time**. Often for real systems, it is important to describe the temporal behaviour of the system. Since the classical net is not capable of handling quantitative time, a timing concept is added. There are many ways to introduce time into the Petri net. Time can be associated to tokens, places and/or transitions.
3. **Extension with hierarchy**. In order to avoid the tendency to large and complex specifications for real systems, a hierarchy construct, called subnet is provided. A subnet is an aggregate of a number of places, transitions, and subsystems. These hierarchies can be used to structure large processes. In the first level a simple description of the process is given, and at the next levels another more detailed behaviours are given.

2.1.2.3 YAWL

YAWL (Yet Another Workflow Language) is a workflow language designed and defined specifically to accomplish the Workflow patterns [vand05]. The language is supported by a software system that includes an execution engine, a graphical editor and a worklist handler. The system is available as an Open source software under the LGPL license [Wiki09].

The language and its supporting system were originally developed by researchers at Eindhoven University of Technology and Queensland University of Technology.

As we have mentioned before, YAWL was born because of the necessity to define a workflow language that would support the Workflow Patterns and that would have a formal semantics. Petri nets was taken as a starting point due to it came close to supporting most of this patterns and it was extended with three main constructs, namely or-join, cancellation sets, and multi-instance activities. These three concepts are aimed at supporting five of the Workflow Patterns that were not directly supported in Petri nets, namely synchronizing merge, discriminator, N-out-of-M join, multiple instance with no a priori runtime knowledge and cancel case. In addition, YAWL adds some syntactical elements to Petri nets in order to capture other workflow patterns such as simple choice like xor-split, simple merge like xor-join, and multiple choice like or-split.

During the design of the language, it turned out that some of the extensions that were added to Petri nets were difficult or even impossible to re-encode back into plain Petri nets. As a result, the original formal semantics of YAWL is defined as a Labelled transition system and not in terms of Petri nets.

Table 2-5 shows us the YAWL notation we can use to define workflow process.

Object	**Description**	**Graphical representation**
Atomic task	Represents a single task to be performed by a human or an external application.	
Condition	Represents a state for the process.	
Input condition	The Input condition is where a process starts.	
Output Condition	The Output condition is where a process ends.	
AND-split	Activates all outgoing links from this task upon completion.	
AND-join	Activates this task when all incoming links have been activated.	
OR-split	Activates a number of outgoing links from this task upon completion.	
OR-join	Activates this task when one or more incoming links are activated and there is no possibility for other links to be activated if the task continues to wait.	
XOR-split	Activates one outgoing link from this task upon completion.	
XOR-join	Activates this task each time an incoming link has been activated.	

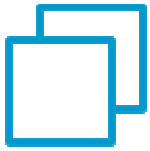
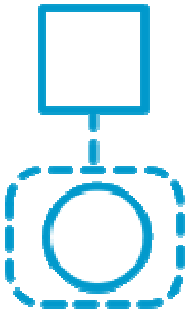
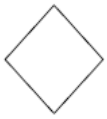
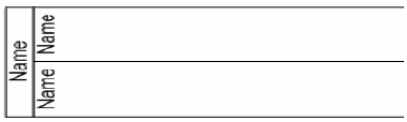
Composite task	The Composite task is a container for another YAWL process, and as such provides decomposition mechanisms.	
Multiple instance task	Allows multiple instances of a task to run concurrently. The minimum and maximum number of instances, the threshold for completion and whether new instances can be created on the fly or not can be specified for this task.	
Cancellation region	In a Cancellation Region, all elements within the dotted region are deactivated upon task activation. Workflow designers can thus specify cancellation of single tasks up to whole processes.	

Table 2-5 YAWL notation [YAWL08]

2.1.2.4 BPMN

The Business Process Modeling Notation (BPMN) is a standard developed by the Business Process Management Initiative (BPMI). The first release of BPMN, the 1.0 specification, was released to the public in May of 2004 and represents more than two years of effort by the BPMI Notation Working Group. The primary goal of the BPMN effort was to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes [Bpmn08].

BPMN defines a Business Process Diagram (BPD), which is based on a flowcharting technique tailored for creating graphical models of business process operations. A BPD is made up of a set of graphical elements that enable the easy development of simple diagrams that will look familiar to most business analysts like a flowchart diagram. The elements were chosen to be distinguishable from each other and to utilize shapes that are familiar to most modellers. For example, activities are rectangles and decisions are diamonds. It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for creating business process models, while at the same time being able to handle the complexity inherent to business processes. The approach taken to handle these two conflicting requirements was to organize the graphical aspects of the notation into specific categories which provide a small set of notation categories, so the reader of a BPD can easily recognize the basic types of elements and understand the diagram. Within the basic categories of elements, additional variation and information can be added to support the requirements for complexity with no more changes in the basic look-and-feel of the diagram. The four basic categories of elements are flow objects, connecting objects, swimlanes and artifacts.

Flow objects	Description	Graphical representation
Event	An Event is represented by a circle and is something that happens during the course of a business process.	
Activity	An Activity is represented by a rounded-corner rectangle and is a generic term for work that company performs.	
Gateway	A Gateway is represented by the familiar diamond shape and is used to control the divergence and convergence of Sequence Flow.	
Connecting objects		
Sequence flow	A Sequence Flow is represented by a solid line with a solid arrowhead and is used to show the order that activities will be performed in a Process.	
Message flow	A Message Flow is represented by a dashed line with an open arrowhead and is used to show the flow of messages between two separate business entities (or business roles) that send and receive them.	
Association	An Association is represented by a dotted line with a line arrowhead and is used to associate data, text, and other Artifacts with flow objects.	
Swimlanes		
Pool	A Pool represents a Participant in a Process.	
Lane	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally.	
Artifacts		
Data object	Data Objects are a mechanism to show how data is required or produced by activities.	
Group	A Group is represented by a rounded corner rectangle drawn with a dashed line. The grouping can be used for documentation or analysis purposes.	

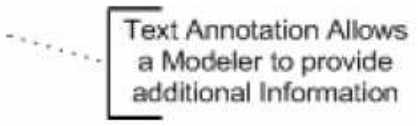
Annotation	Annotations are a mechanism for a modeler to provide additional text information for the reader of a BPMN Diagram.	

Table 2-6 BPMN syntax notation

Now we can see a process about a request approval defined using the BPMN syntax notation we have explained before (see Figure 2-6).

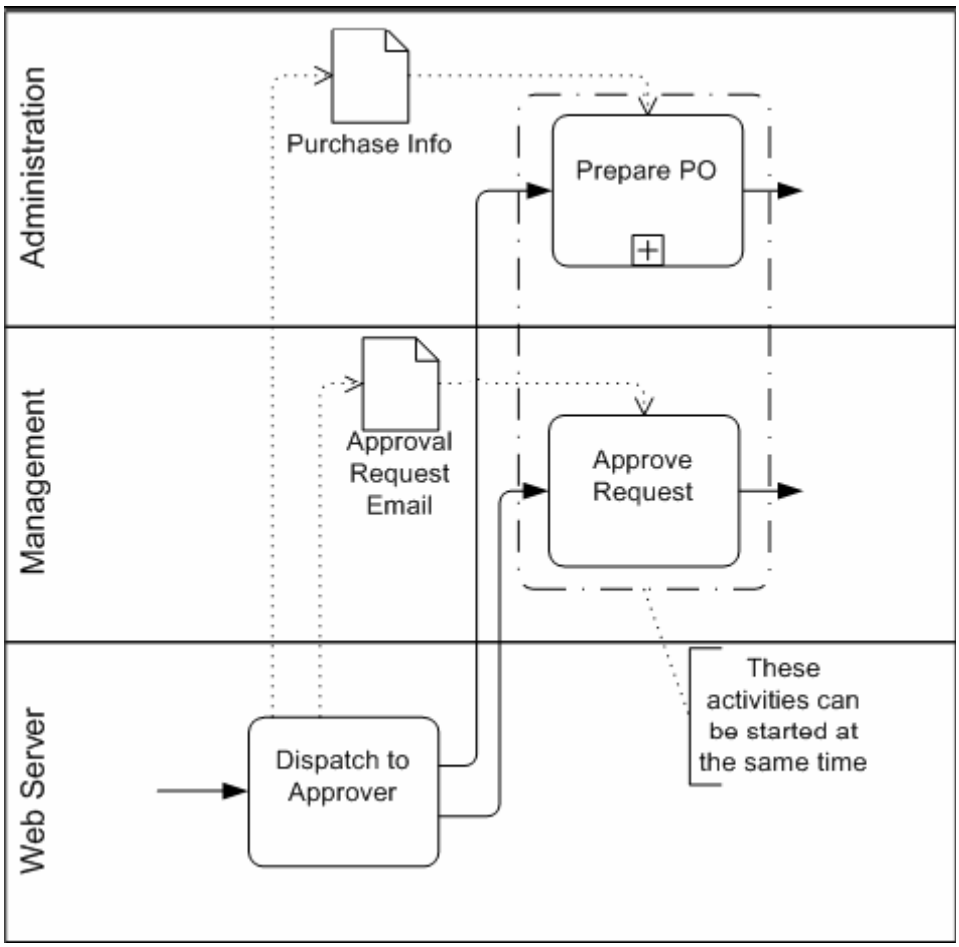


Figure 2-6 A Segment of a Process with Data Objects, Groups, and Annotations

2.1.2.5 BPEL

Business Process Execution Language for Web Services (BPEL or BPEL4WS) is a language used for the definition and execution of business processes using Web services. BPEL enables the top-down realization of Service Oriented Architecture through composition, orchestration, and coordination of Web services. BPEL provides a relatively easy and straightforward way to compose several Web services into business processes [Orac09].

BPEL builds on the foundation of XML and Web services; it uses an XML-based language that supports the Web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination, and WS-Transaction.

BPEL represents a convergence of two early workflow languages; Web Services Flow Language (WSFL) and XLANG. WSFL was designed by IBM and is based on the concept of

directed graphs. XLANG, a block-structured language, was designed by Microsoft. BPEL combines both approaches and provides a rich vocabulary for description of business processes.

The first version of BPEL was developed in August 2002. In April 2003, BPEL was submitted to the Organization for the Advancement of Structured Information Standards (OASIS) for standardization purposes, and the Web Services Business Process Execution Language Technical Committee (WSBPEL TC) was formed.

A BPEL process specifies the exact order in which participating Web services should be invoked, either sequentially or in parallel. With BPEL, you can express conditional behaviours. For example, an invocation of a Web service can depend on the value of a previous invocation. You can also construct loops, declare variables, copy and assign values, define fault handlers, etc. By combining all these constructs, BPEL lets us to define complex business processes in an algorithmic manner.

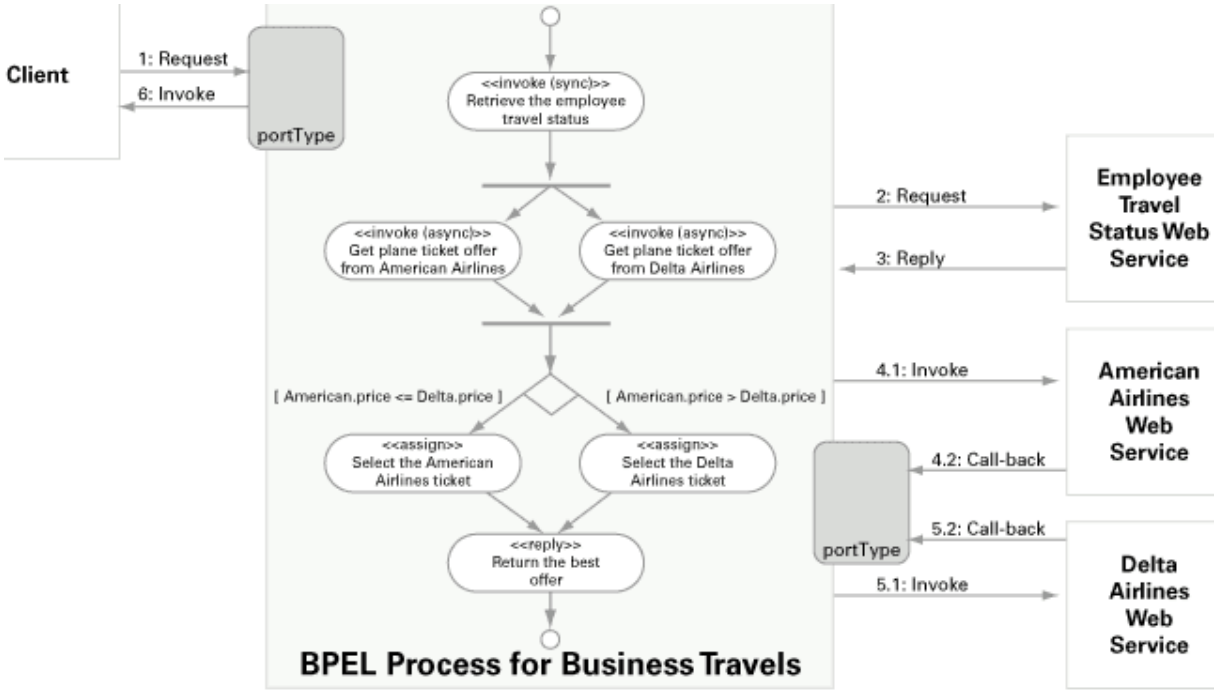


Figure 2-7 Example BPEL process for travel arrangements

An example of a BPEL process defined for a travel agency is showed in the Figure 2-7.

2.2 Workflow tools

The former section introduced the most popular workflow notations used to define business process. In this section, we are going to know some software systems which use these notations or are based in it. We will start with a brief selection of commercial software and after we will define the two applications we are going to use in the development.

There are some workflow concepts that are common in many systems [Mano02]:

1. **Monitoring**, used for contributing information about the circumstances of workflow during execution.
2. **History** of workflow actions for evaluation or recovery.

3. **Persistence** to save the historic information and provide access to it.
4. **Manual Intervention** for changing the order that activities are performed in as they are performed.
5. **Worklist** to coordinate the activities among the workers.
6. **Federated Workflow** to address the issue of how workflow systems interoperate.

2.2.1 Commercial software

The commercial software chosen to introduce briefly in the next pages are Staffware, Websphere MQ Workflow, The Progression Model, Microsoft Windows Workflow Foundation, Flexo Business and Business Process Visual Architect [Guer06].

2.2.1.1 Staffware

Staffware is a workflow management system developed by the Staffware company from England. Apart from creating a seamless bridge between the different areas of the system, Staffware also manages the processes, automatically routing specific tasks to the appropriate person in the right part of the organizational structure [Tibc09]. This commercial software supports the BPMN notation.

Its process definition tool is the Graphical Workflow Definer (GWD, see Figure 2-8). GWD has the advantage of being very clear in visual terms. Using Staffware, the workflow manager can define three different kinds of tasks: automatic, normal and eventual tasks.

The resource management allows defining different user groups. Each user has an agenda where they have the work items allocated to them, and the allocation is done to the entire groups we have mentioned [Lema07].

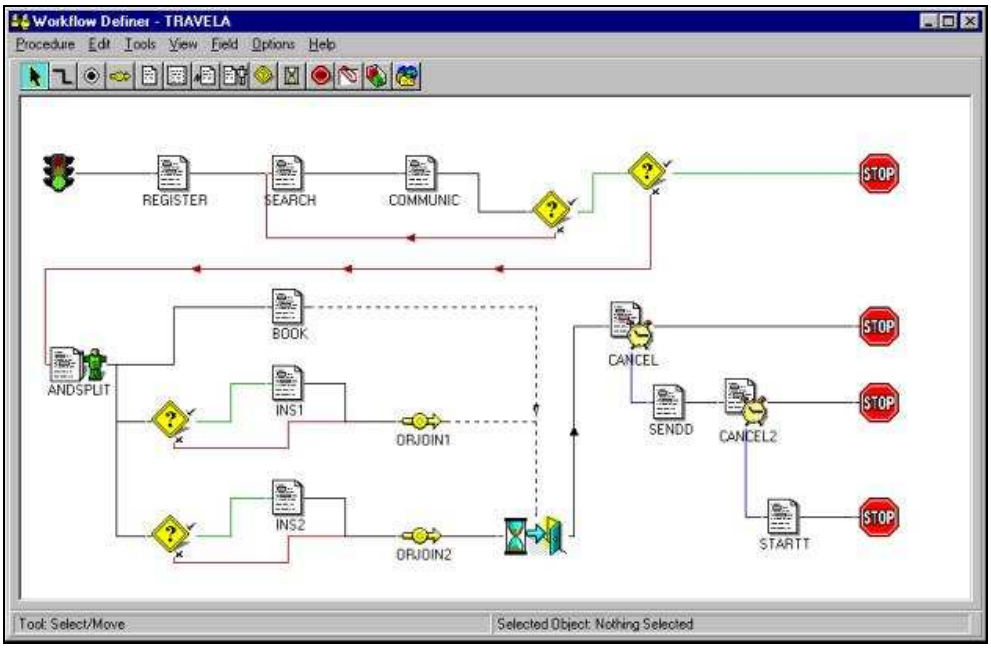


Figure 2-8 Graphical Workflow Definer of Staffware

2.2.1.2 Websphere MQ Workflow

WebSphere MQ Workflow supports long-running business process workflows as they interact with systems and people. Automates and tracks business processes in accordance with business design. Provides integration processes with rich support for human interactions. Enables use with WebSphere Business Integration Modeler and Monitor for design, analysis, simulation and monitoring of process improvements. Buildtime is the graphical process definition tool that is part of the WebSphere MQ Workflow product. You can graphically define business processes and their activities to the level of detail needed for automation. Buildtime includes graphical support for declaring and documenting:

1. business rules on process navigation between steps,
2. business rules for role-based work assignment,
3. process interface definitions (data, programs, queues).

When you first start Buildtime, you see the Buildtime window (see Figure 2-9). There is a tree view on the left of the Buildtime window that shows all the objects that belong to workflow models. The tabs at the top of the tree view provide a quick way to switch between the different trees. The tabs indicate that you can display object trees for Processes, Staff, Network, and Implementations. The right part of the Buildtime window is a work area that is used to display views of workflow elements. This can be the diagram view of a process or the properties that you can define for a selected object. At the bottom of the Buildtime window, there is a Status bar. The status bar shows information such as the name of the database you are using and your user ID.

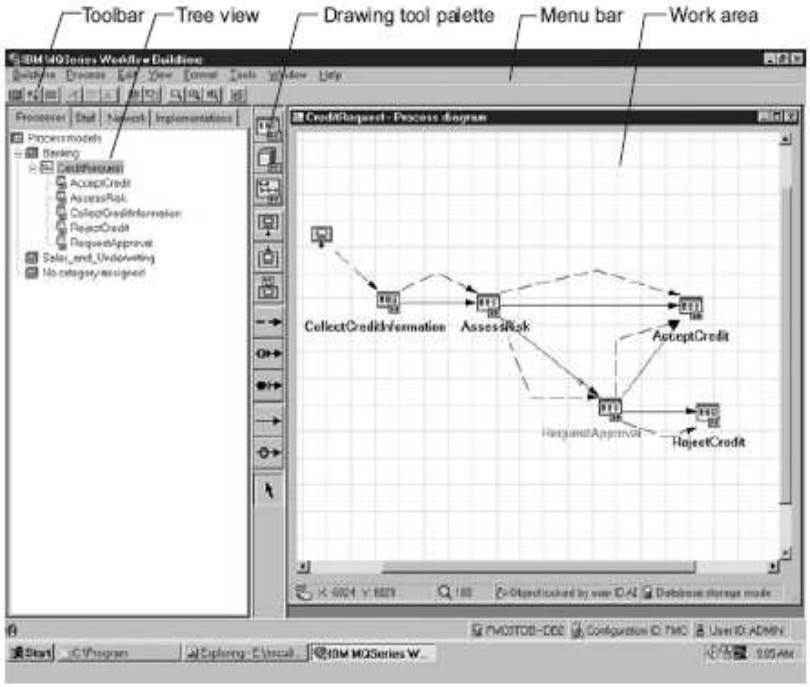


Figure 2-9 Buildtime user interface

2.2.1.3 The Progression Model

The Progression model [Stav04] has incorporated some of the managing concepts of workflow to increase the flexibility in IS. It makes explicitly the steps and transactions as user undertakes when using an IS. As the user progresses towards accomplishing a task or goal,

the progression model infrastructure records each step and the state of the transaction and workflow. A progression is a workflow transaction and a sequence of scenes in a process to create a workflow transaction. The progression model is displayed to the user in the single progression section through four main panels:

1. **User interface**. This panel displays the widgets or interface elements that are specified in the user interface element of the markup document for the current scene.
2. **Transaction**. This panel displays all the transaction items of the progression as specified in the workflow transaction element of the markup document, including the information that the user has entered up to that point in the progression.
3. **Workflow**. This panel displays the status information for the progression as specified in the workflow element of the markup document. The workflow status shows the scenes of the progression in the recommended order. Each scene is associated with a name, the worker assigned to complete it, and the current status of its completion; these are all displayed in a table format.
4. **Feedback**. This panel consists of constraints, information status, and markup document. The constraints sub-panel display any information related to constraints that are violated throughout the progression. The information status sub-panel display the information that is missing and required to complete the transaction. The markup document sub-panel displays the current state of the markup document. The single section also has buttons to start a new progression, open an existing progression, save the current progression, and quit the progression analyzer.

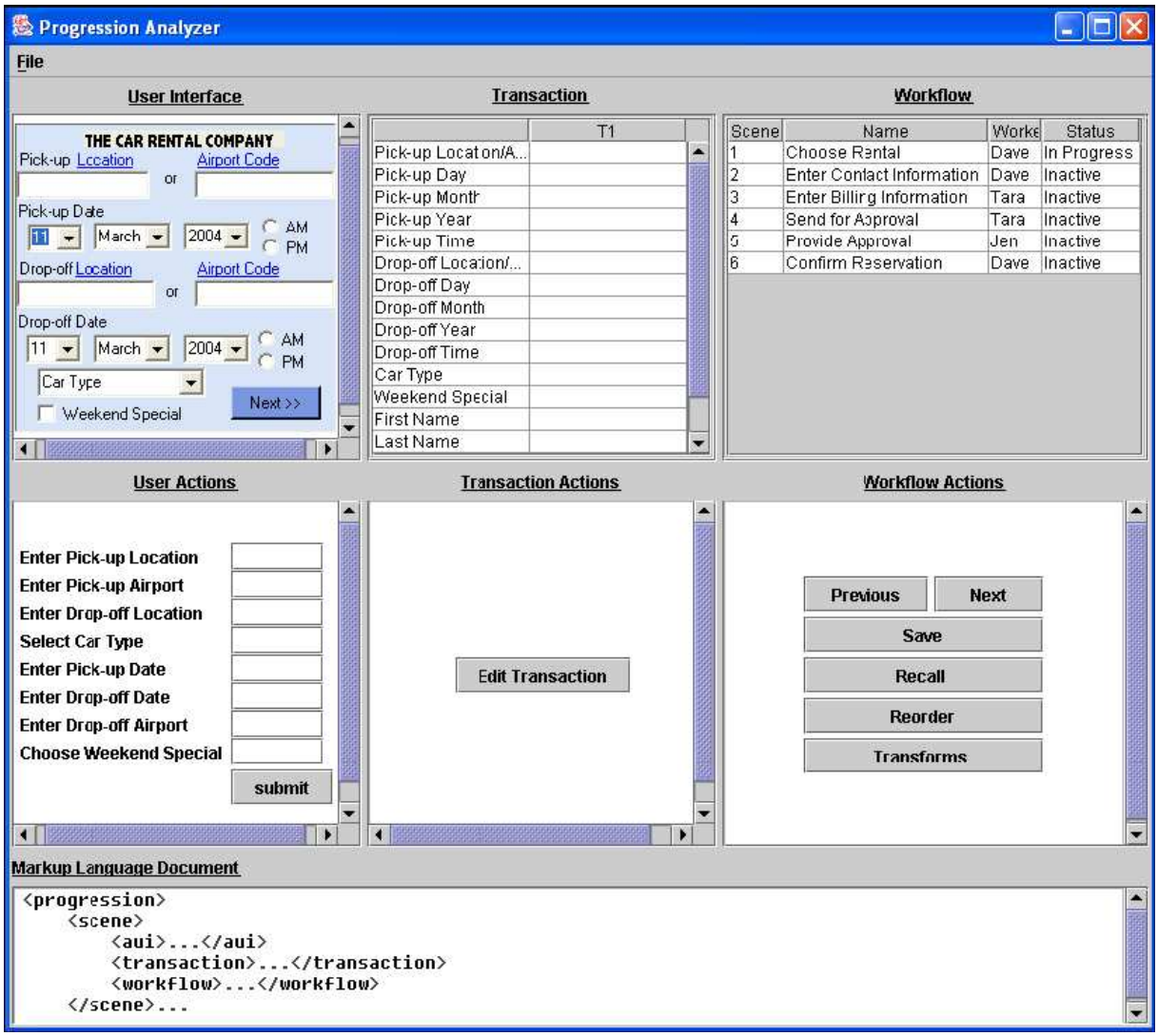


Figure 2-10 The progression analyzer displaying the Enter Name scene

2.2.1.4 Microsoft Windows Workflow Foundation

Microsoft Windows Workflow Foundation (WWF) [Espo05] is an extensible framework for developing workflow solutions on the Windows platform. It provides a single, unified model to create end-to-end solutions that span categories of applications, including human workflow and system workflow. WWF supports two fundamental workflow styles: sequential workflow and state machine workflow. A sequential workflow is useful for operations expressed by a pipeline if steps that execute one after the next until the last activity completes. Sequential workflows are not purely sequential in their execution, they can still receive external events or start parallel tasks, in which case the exact sequence of execution can vary somewhat. A state machine workflow is made up of a set of states, transitions and actions. One state is denoted as a start state, and then based on an event a transition can be made to another state. The state machine workflow can have a final state that determines the end of the workflow.

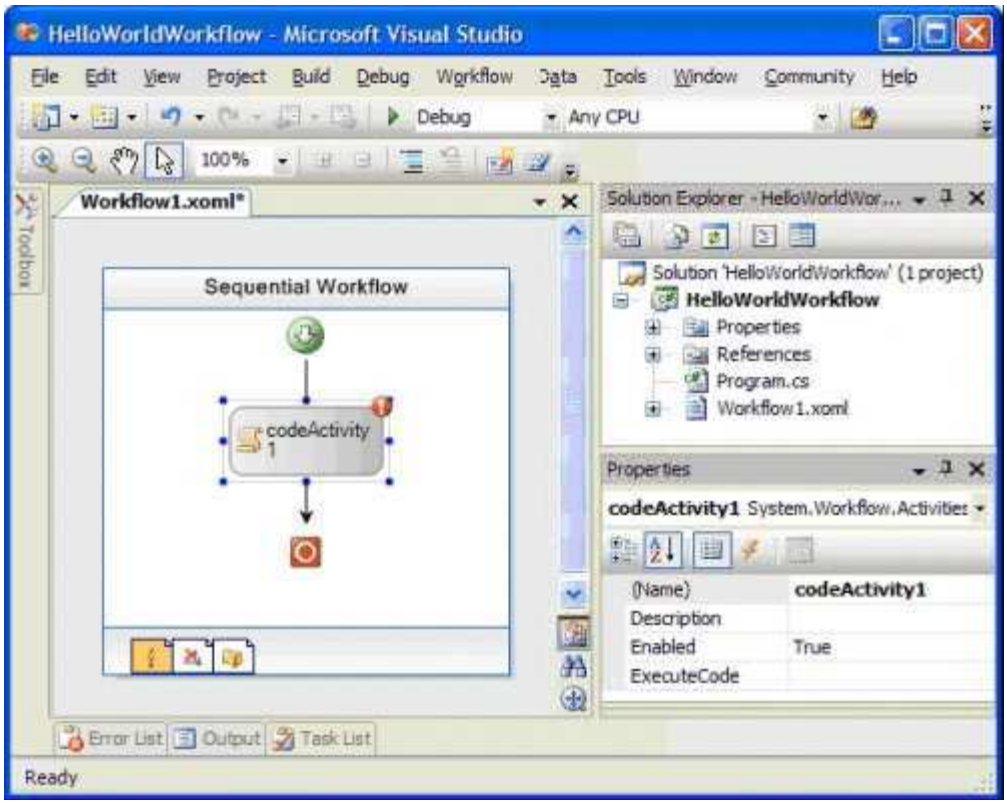


Figure 2-12 The Visual Studio 2005 Workflow designer

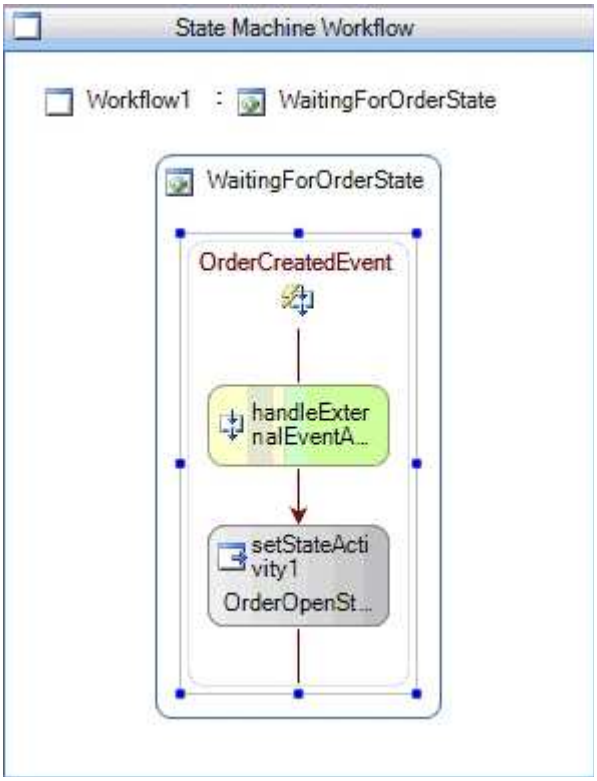


Figure 2-13 An interior view of the OrderCreatedEvent EventDriven activity

2.2.1.5 Flexo business

The Flexo Business [Dena08] is a Workflow Management System based on Petri nets formalism for process description. It interprets the workflow description, controls the

instantiation of processes, sequences activities, adds work items to the user work lists and invokes application tools when necessary. Flexo Business is an extremely powerful yet simple tool that facilitates the collaborative definition of applications by modelling underlying processes, workflows, and graphical interfaces together with external connections to databases and existing services. The Flexo Workflow Engine interprets the Flexo model, enabling it to automatically run an application that interacts with databases and external services. Flexo Workflow Engine is also BPEL compatible (Business Process Execution Language) and therefore can be used in relation with other BPEL engines.

Using intuitive and familiar drag-and-drop techniques from a palette of pre-configured items, business analysts can define the underlying work distribution and navigational pages of the application in seconds.

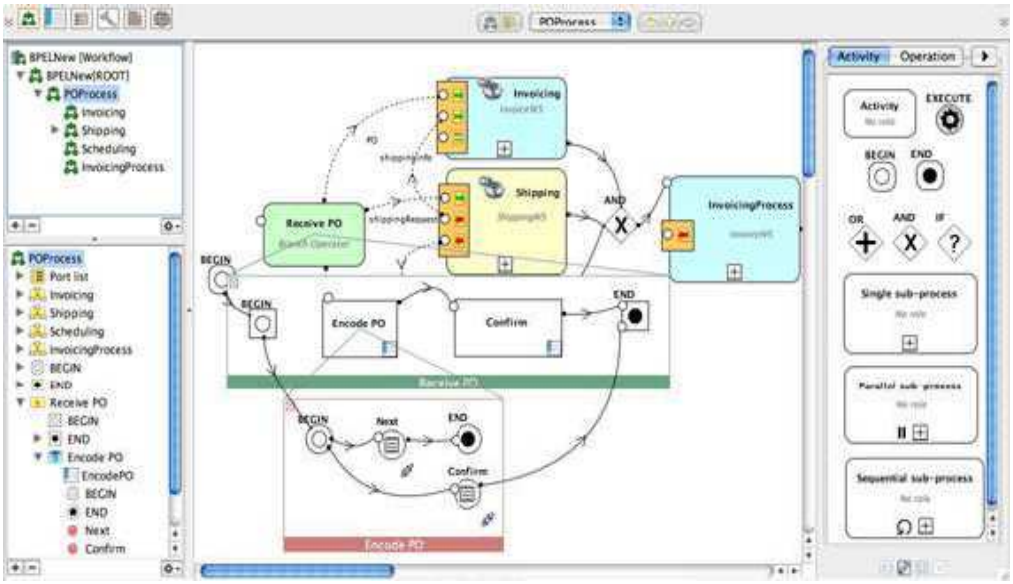


Figure 2-14 The workflow modeller [Dena08]

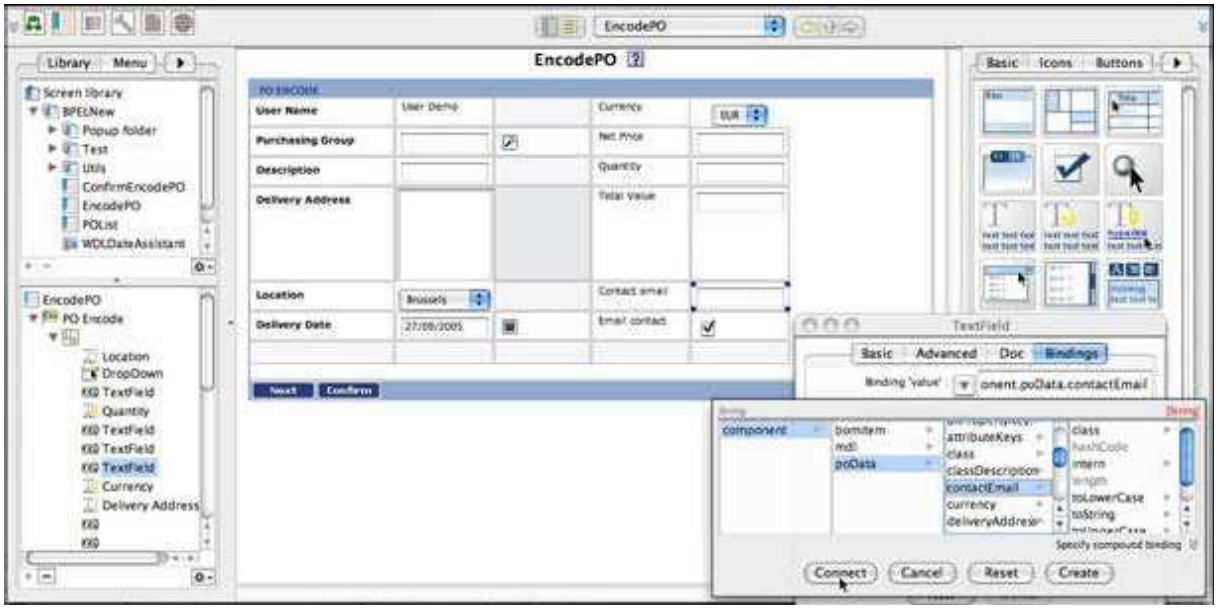


Figure 2-15 Layout of user graphical interfaces editor

Pages and forms are easily constructed by dragging/dropping texts, images and preconfigured interaction widgets (entry fields, drop-down lists, etc.) to their desired location. The addition

of realistic data samples to the pages and the forms thus defined bring a first round of simple prototypes to life ready for improvement.

2.2.1.6 Business Process Visual Architect

Business Process Visual ARCHITEC (BP-VA) is a visual modelling tool that provides the most extensive support for the Business Process Modeling Notation (BPMN). BP-VA adopts the resource-centric interface, where context-sensitive shortcut buttons will be shown around the active diagram element. Each resource provides a functionality that you would likely to perform frequently, like creating a connection to a new/existing shape, opening the model specification, resizing a shape to fit.

BP-VA completely covers the BPMN, from model specification to graphical notation, including different presentation options.

The Diagram pane is a tabbed view of all opened diagrams. You can click on the tab of a diagram to make it the active diagram for viewing or editing.

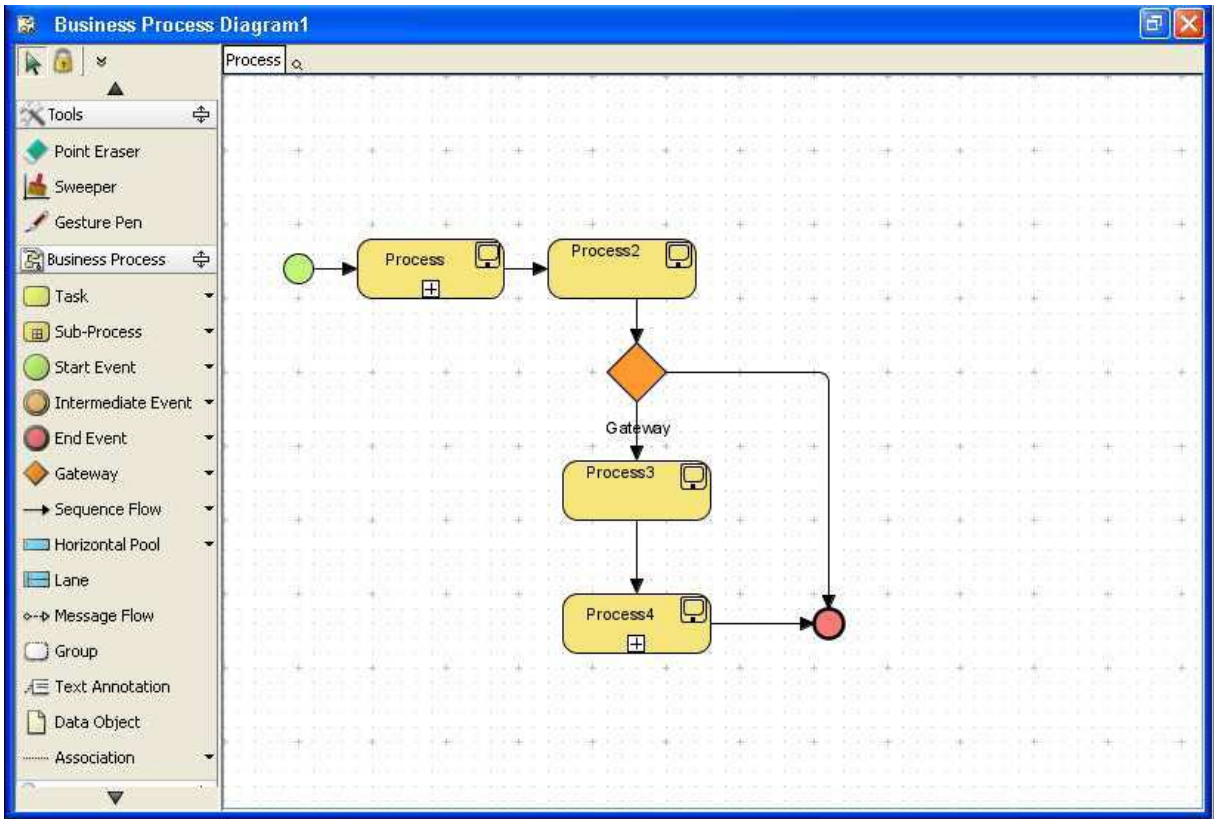


Figure 2-16 User interface of BP-VA

2.2.2 IMmedia S.A. Atoms

Atoms, a product from IMmedia S.A., is distributed like an information management system that allows the user, among other things, to define workflow process using its workflow editor. This commercial workflow software is the one we are going to work with. It is for that reason it is out of the former section about commercial software.

It is built using Java, XML, XSL, XHTML and CSS technologies:

1. **XML** to store data base format data.
2. **Java** to develop the entire application layer and to create the last mentioned XML files.
3. **XSL** to interpret and transform the XML files to obtain XHTML pages.
4. **XHTML** pages that a web browser can execute.
5. **CSS** with a defined style attached to each XHTML page.

Atoms is a software that allows users an integrated management of all company's information through the different modules it integrates, like content management, document management, customer relationship management, learning management, etc.

Going depth in its workflow editor, the tool we are going to use primarily, it allows users to define business processes, adding concepts to the different transitions (tasks) like time to execute or to be valid, and execution in charge of the server or humans, attach forms to the different tasks, etc. Its workflow editor works using UML State charts notation, but it does not accomplish this notation strictly.

Using its workflow editor we ca define states and transitions, the main objects to run a process there. We can also expand a state to include more states inside it, but it will be only a graphical feature that can make easier the comprehension of the process. Moreover, we can define synchronization bars which should allow defining two different ways to follow, but it is not possible (it is not possible yet, maybe in a new version).

To run a process, this application needs to create a document first, which will store the different information created throughout the course of the process. This document will be placed in the actual state of the process execution and it can only be unique. Then, it would not be possible to have two different tasks in execution at the same time.

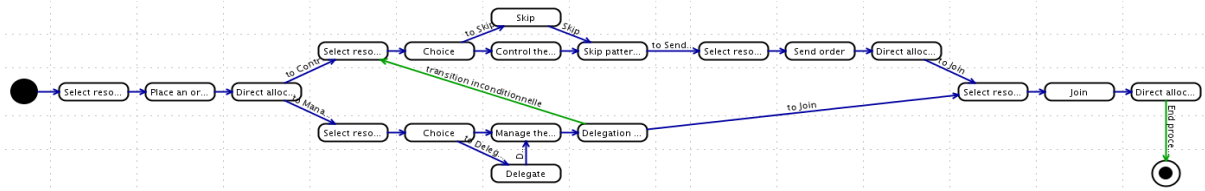


Figure 2-17 A screenshot of the Atoms workflow editor

Figure 2-17 shows us how a process can be defined in the workflow editor of Atoms last mentioned.

2.2.3 FlowiXML Workflow editor

FlowiXML workflow editor is a tool developed at Université Catholique de Louvain by Christophe Lemaigre [Lema07]. This tool allows us defining workflow processes using petri nets notation, and in terms of the workflow resource patterns, it allows applying this kind of resources to each task in its definition.

Defining the workflow process, the tool allows us to add places, tasks, organizational units and resource boxes:

1. **Places and tasks**. They are used to represent the process in petri nets notation.

2. **Organizational units**. We can define organizational units to represent the different departments of the organization and to classify these different tasks in these departments. In Figure 2-18 we can see the organizational units represented in the boxes called Stock department, Delivery department and Accountant department.
3. **Resource boxes**. We can define user stereotypes and jobs and represent them using resource boxes in the process. In Figure 2-18 we can see a resource box represented called Warehouseman.

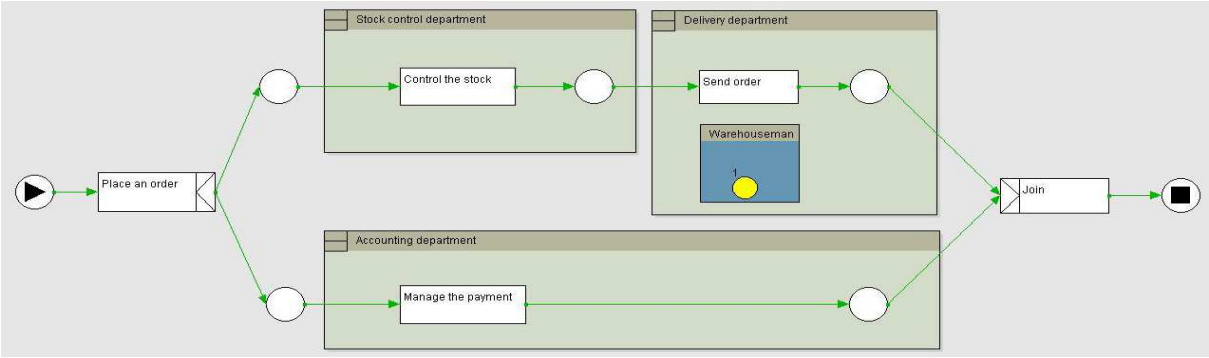


Figure 2-18 A screenshot of the FlowiXML workflow editor

Figure 2-18 shows us a screenshot of a process defined using FlowiXML workflow editor. Concretely, this process represents the process of the first and simplest case we are going to study (see section 5.1).

To add these different objects the application uses a very understandable toolbar in the left side, using the right side to show the different options to configure each object added (see Figure 2-20). However, to create the different user stereotypes and jobs we need to go to the main user interface of the application and select the options to edit them (see Figure 2-19).

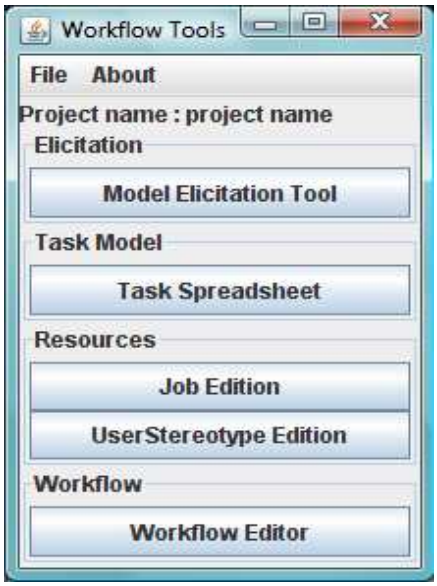


Figure 2-19 The main user interface

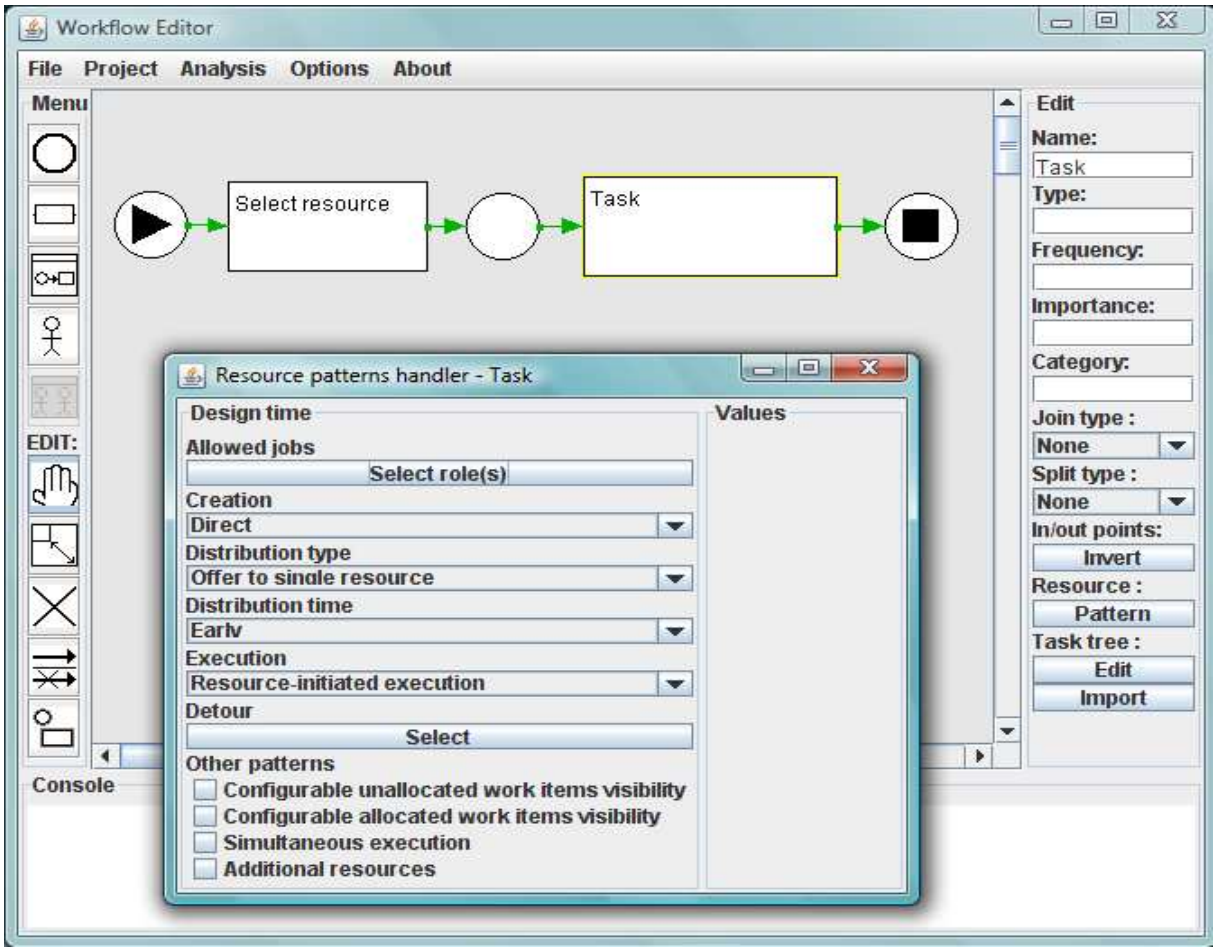


Figure 2-20 Workflow editor user interface

As we can see in the workflow editor user interface (Figure 2-20), inside the allowed configuration options in a task, we can specify a pattern clicking the button called pattern labelled with resource. After clicking it, the tool shows us the window resource pattern handler as we can see in the picture, allowing the definition of the different workflow resource patterns of creation, distribution type, distribution time, execution and detour.

Moreover, FlowiXML workflow editor allows us to export the workflow information of our project in an XML file, whose specification can be seen in further chapters (see section 4.3.1).

Chapter 3 eXtensible Mark-up Languages for User Interface Definitions

XML has become one of the most used languages in programming. Using XML, you can store structured data in an external file, design web pages that any device could read using derivatives like XHTML… and design user interfaces in a low level, amongst other uses.

Nowadays, we can visit a website since our laptop at the office, using a cell phone while walking, or using one of the new video game consoles which includes web browser software. Each device has different hardware interfaces, bigger or smaller screens, keyboard or only eight or nine buttons…

Thanks to XML, we can define abstract user interfaces to adapt it later depending on the device that is going to show it.

In this chapter, we start introducing the XML language to show later how we can represent a user interface using it and how we can read and transform it.

3.1 Introduction to XML

XML is, in fact, a very simple technology which has around it other technologies that complement and make it bigger and with major possibilities. Together with all related technologies, XML represents a different way to work, more advanced, with the information in software applications, whose principal innovation consists in allowing sharing the information not in a concrete level or to a specific environment, but for all the environments and supports.

Then, XML plays an important role in the actual world, which tends to the globalization and the compatibility among the systems, since the technology is the one which will allow sharing the information in a sure, trustworthy and easy way. In addition, XML allows the programmer to dedicate his efforts to the important tasks when he works with the information, since some tedious tasks like data validation or data searching in the XML structure is chargeable to the language and it is specified by the standard, so that the programmer does not have to worry about that.

There is a world of technologies about XML, which provide easier and interesting ways to work with the information and, definitively, an advance at the moment of treating the information, which is indeed the aim of the computer science in general. So, the world of XML is not a language nor syntax, but several of both, and is not a totally a new way to work, but a more refined one.

The World Wide Web Consortium (W3C) defines XML as follows [W3C09]:

"Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere."

Now, we can see an example of how the data is stored in a XML file (see Figure 3-1).

```
<bookstore>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
  <book category="PHILOSOPHY">
    <title lang="en">Tao Te Ching</title>
    <author>Lao Tse</author>
    <year>1993</year>
    <price>19.95</price>
  </book>
</bookstore>
```

Figure 3-1 An example of a bookstore definition in a XML file

In figure 3-1, we can see how the data is represented in a XML file. After defining the bookstore, we have stored two of its books, saving the information of title, author, year of publication and its price.

In conclusion, XML is a markup language that allows users to define a set of tags that define the structure of a document. While XML can help in exchange of semantic information, it still

lacks routing information. Such information is critical to enable proper routing of a document within and across organizations [vand01].

3.2 User Interface Description Languages based on XML

XML is frequently used in user interface design. A User Interface Description Language (UIDL) consists of a high-level computer language for describing characteristics of interest of a user interface with respect to the rest of an interactive application. It helps define user interfaces linguistically with a general trend to do so in an XML-complaint way. Many UIDLs have been conceived that contain different features and focus on different levels of granularity [Guer05]. Many XML-compliant languages for defining user interfaces have identified and analyzed [Souc03].

The following definitions have been extracted from Conceptual Modeling of User Interfaces to Workflow Information Systems by Josefina Guerrero García [Guer06].

3.2.1 XIML

XIML (eXtensible Interface Mark-up Language) affords the ability to describe a user interface without concern for the implementation [Puer02]. XIML is functional across the entire lifecycle of UI: design, development, operation, management, organization, and evaluation. It is able to relate the abstract and concrete data elements of an interface.
XIML defines five basics interface elements: (1) the task component that captures the business process and/or user tasks that the interface supports; (2) the domain component which is a set of all the objects and classes used; (3) the user component that captures the characteristics of the users that can use the application; (4) the dialog component that determines the UI interaction, and (5) the presentation component. Additionally, XIML includes attributes and relations to connect with the elements.

The main concern of the XIML approach is the model-based development itself. It provides a standard mechanism to data interchange among tools and application from design to operation. One shortcoming of XIML is that only graphical user interfaces are supported. The language is used by industry in commercial products; however XIML is available via a non-commercial research license. XIML do contain a mechanism for specifying any general-purpose model: this could be used to specify a workflow model, but the designer is entirely left without any conceptual and methodological guidance.

3.2.2 UIML

The UIML (User Interface Mark-up Language) allows the user to specify the user interface in general terms and then render it according to a style description [Abra99]. It was designed conforming to XML and has HTML-like syntax. A UIML document contains three different parts: 1) a UI description, 2) a peers section that defines mappings from the UIML document to external entities, and finally 3) a template section that allows the reuse of already written elements. This language allows the designer to specify the appearance, user interaction, and application connection of the user interface. UIML is also independent of any user interface metaphor, such as graphical user interfaces or voice-response, but not multimodal.

3.2.3 XUL

XUL (XML-based User-Interface Language) is a multi-platform language to describe application UI. The most of GUI components could be create using XUL – buttons, text-boxes, check-boxes, menus, dialog boxes, trees and others [Bosw02].

XUL is similar to the Java approach but it uses the Mozilla engine instead of the Java Virtual Machine. It is the ideal solution to Web System when it is not possible to install a standard browser. It is also possible to design on-line and off-line application. XUL has its focus on window-based graphical user interfaces. The main disadvantage is the low portability because it cannot be used in every browser [Gome04]. XUL could be considered mainly as a UI markup language for browsers, like HTML is for web pages. It does not represent a specification of this user interface connected with other aspects, like the task model.

3.2.4 AUIML

AUIML (Abstract User Interface Mark-up Language) is "an XML vocabulary which has been designed to allow the intent of an interaction with a user to be defined" [Azev00]. This clearly contrasts with the conventional approach to user interface design, which focuses on appearance. It is intended to allow designers to focus on the semantics of the interactions rather than the particular devices that need to be supported. Being an XML vocabulary, AUIML allows device independent encoding of information. All the interaction information can be encoded once and subsequently rendered using 'device dependent rendering' so that users can actually interact with the system. AUIML is therefore intended to be independent of the client platform on which the user interface is rendered, the implementation language and the user interface implementation technology [Gome04]. As AUIML is mostly developed for internal use at IBM, most information is confidential, thus limiting the usage of this UIDL outside.

3.2.5 UsiXML

UsiXML (USer Interface eXtensible Mark-up Language) is a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. In other words, interactive applications with different types of interaction techniques, modalities of use, and computing platforms can be described in a way that preserves the design independently from peculiar characteristics of physical computing platform. UsiXML is defined in a set of XML schemas. Each schema corresponds to one of the models in the scope of the language. UsiXML consists of a User Interface Description Language (UIDL) that is a declarative language capturing the essence of what a UI is or should be independently of physical characteristics. It describes at a high level of abstraction the constituting elements of the UI of an application: widgets, controls, containers, modalities, interaction techniques, etc. UsiXML allows cross-toolkit development of interactive application. A UI of any UsiXML-compliant application runs in all toolkits that implement it: compilers and interpreters.

UsiXML supports device independence: a UI can be described in a way that remains autonomous with respect to the devices used in the interactions such as mouse, screen, keyboard, voice recognition system, etc. In case of need, a reference to a particular device can be incorporated. UsiXML supports platform independence: a UI can be described in a way

that remains autonomous with respect to the various computing platforms, such as mobile phone, Pocket PC, Tablet PC, laptop, desktop, etc. In case of need, a reference to a particular computing platform can be incorporated.

UsiXML supports modality independence: a UI can be described in a way that remains independent of any interaction modality such as graphical interaction, vocal interaction, 3D interaction, or haptic. It allows reuse of elements previously described in anterior UIs to compose a UI in new applications..." [UsiX08].

3.2.6 User Interfaces definition in IMmedia S.A. Atoms

In this case, the application where we are going to generate automatically the user interfaces do not use any of the past proposals. This application has a non-standard definition of user interfaces, only valid for it, and generated directly from java source code. For this reasons, the only way to create a user interface for a given task is using the form editor that Atoms includes in its environment.

3.3 XML transformations

Last section showed us some ways to define user interfaces. Now, we are going to show how to transform these user interface definitions or any other kind of XML definition (as we sill see in further chapters) into readable content of our necessity.

There are some technologies we can use to modify a XML file. The former chapters have already introduced some concepts like workflow and user interface. The two workflow system editors we are going to use in this thesis store their workflow processes information in a XML file. We need to transform this information from one of these editors to another, and as following, we present what technologies we could use to do it.

3.3.1 XSLT Stylesheet Language

XSLT (eXtensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents.

XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL [W3C09] [W3Sc09].

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>Available books</h2>
    <table border="1">
      <tr>
        <th>Title</th>
        <th>Author</th>
      </tr>
      <xsl:for-each select="bookstore/book">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="author"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Figure 3-2 An example of XSLT stylesheet to transform the last XML file definition (Figure 3-1) in a HTML webpage

Available books

Title	Author
Learning XML	Erik T. Ray
Tao Te Ching	Lao Tse

Figure 3-3 The output obtained after apply our XSLT stylesheet (Figure 3-2) to the last XML file definition (Figure 3-1)

In Figure 3-2 we can see a XSLT stylesheet which we can apply to the XML file we have defined before (see Figure 3-1). The `<xsl:template>` declaration is used to build a template for an element defined by its attribute match, which is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document using `match="/"` like in our example.

The `<xsl:for-each>` element can be used to select every XML element of a specified set of nodes. Using the attribute `select="bookstore/book"` we select all the books defined in our bookstore. Iterating each book, using the `<xsl:value-of>` element we can extract the value of an XML element and add it to the output stream of the transformation. In our example, using the `select="title"` attribute we extract the title information of the actual book iterated.

3.3.2 Transforming XML with Java

Java is one of the best languages for transforming XML documents. Its strong Unicode support in particular made it the preferred language for many early implementers. Consequently, more XML tools have been written in Java than in any other language. More

open source XML tools are written in Java than in any other language. More programmers process XML in Java than in any other language.

As following, we present the three major standard APIs for processing XML documents with Java, the Simple API for XML (SAX), the Document Object Model (DOM), Java API for XML Processing (JAXP) [Cui05].

3.3.2.1 SAX

SAX, the Simple API for XML, was the first standard API shared across different XML parsers. SAX is unique among XML APIs in that it models the parser rather than the document. In particular the parser is represented as an instance of the XMLReader interface. The specific class that implements this interface varies from parser to parser. Most of the time you only access it through the common methods of the XMLReader interface.

A parser reads a document from beginning to end. As it does so it encounters start-tags, end-tags, text, comments, processing instructions, and more. Parsing is the process of reading an XML document and reporting its content to a client application while checking the document for well-formedness.

SAX represents parsers as instances of the XMLReader interface. The parser tells the client application what it sees as it sees it by invoking methods in a ContentHandler object. ContentHandler is an interface the client application implements to receive notification of document content. The client application will instantiate a client-specific instance of the ContentHandler interface and register it with the XMLReader that going to parse the document. As the reader reads the document, it calls back to the methods in the registered ContentHandler object.

3.3.2.2 DOM

The Document Object Model, DOM, provides a standard set of objects for representing HTML and XML documents, and a standard interface for accessing and manipulating them. It is the second major standard API for XML parsers. Most major parsers implement both SAX and DOM. DOM programs start off similarly to SAX programs, by having a parser object read an XML document from an input stream or other source. However, where the SAX parser returns the document broken up into a series of small pieces, the equivalent DOM method returns an entire Document object that contains everything in the original XML document. One can read information from the document by invoking methods on this Document object or on the other objects it contains. This makes DOM much more convenient when random access to widely separated parts of the original document is required.

The DOM is separated into different parts (Core, XML, and HTML): (1) Core DOM - defines a standard set of objects for any structured document, (2) XML DOM - defines a standard set of objects for XML documents, (3) HTML DOM - defines a standard set of objects for HTML documents.

The XML DOM views XML documents as a tree structure of elements embedded within other elements. All elements, their containing text and their attributes, can be accessed through the DOM tree. Their contents can be modified or deleted, and new elements can be created by the DOM. The elements, their text, and their attributes are all known as nodes.

3.3.2.3 JAXP

Starting in Java 1.4, Sun bundled the Crimson XML parser and the SAX2, DOM2, and TrAX APIs into the standard Java class library. (TrAX is an XSLT API that sits on top of XML APIs like SAX and DOM.) They also threw in a couple of factory classes, and called the whole thing the “Java API for XML Processing” (JAXP).

The reason about include the JAXP to the XML processing API is explained as following:
DOM represents a document tree fully held in memory. It is a large API designed to perform almost every conceivable XML task. It also must have the same API across multiple languages. Because of those constraints, DOM does not always come naturally to Java developers who expect typical Java capabilities such as method overloading, the use of standard Java object types, and simple set and get methods. DOM also requires lots of processing power and memory, making it intractable for many lightweight Web applications and programs.

SAX does not hold a document tree in memory. Instead, it presents a view of the document as a sequence of events. For example, it reports every time it encounters a begin tag and an end tag. That approach makes it a lightweight API that is good for fast reading. However, the event-view of a document is not intuitive to many of today's server-side, object oriented Java developers. SAX also does not support modifying the document, nor does it allow random access to the document.

JAXP attempts to incorporate the best of DOM and SAX. It's a lightweight API designed to perform quickly in a small-memory footprint. JAXP also provides a full document view with random access but, surprisingly, it does not require the entire document to be in memory. The API allows for future flyweight implementations that load information only when needed. Additionally, JAXP supports easy document modification through standard constructors and normal set methods.

3.3.3 Composition of our solution

We are going to transform the workflow process from the FlowiXML workflow editor to the Atoms Defimedia workflow process specification, using a XSLT stylesheet, which it will be called from Java code. In the next chapter you will be able to see the entire stylesheet used.

Chapter 4 Development of a user interface generator for a workflow information system

At this point, we know some technologies used to transform XML files and some notations used to define business processes. We only need to know what are the workflow resource patterns to understand how to solve the main problem we deal with in this thesis.

In this chapter, we will start to introduce the approach of the problem and the solution proposed. Afterwards, we will define all workflow resource patterns, defining the representation in terms of user interfaces we have chosen for each one.

Once we have decided how to represent each pattern, we will show the way to store the information in the two application we will work with, FlowiXML workflow editor as source application and IMmedia Atoms as target application. Then, we will show the transformation done to obtain a user interface for each workflow resource pattern in the target application applied in the source application.

4.1 Overview of the global approach

As we have mentioned in former chapters, the main goal of this thesis is to transform the output of a workflow editor to be used by a workflow management system. This workflow editor allows defining workflow processes in petri nets notations and it lets the definition of the workflow resource patterns for each task in these processes. However, the target application does not allow us to define this kind of patterns, so in this transformation is necessary to reflect the behaviour of the applied pattern in the task execution.

How we will transform each of these patterns is explained in the next section, but we are going to show an example of one of these transformations first. In this example, we will show the transformation of the simplest pattern, which it is called direct allocation and it represent a direct resource assignment to a task.

Imagine we have defined a task in FlowiXML workflow editor (see Figure 4-1) and we decide to apply the mentioned pattern direct allocation.

Figure 4-1 Task represented in FlowiXML workflow editor

If we have to represent that in a workflow information system that does not allow us to use this pattern, we have to define the behaviour of this pattern (see Figure 4-2). In this case, to represent a direct resource assignment to a task we only have to add another task before the execution of the main task that allows the workflow manager to select the resource which is going to execute it.

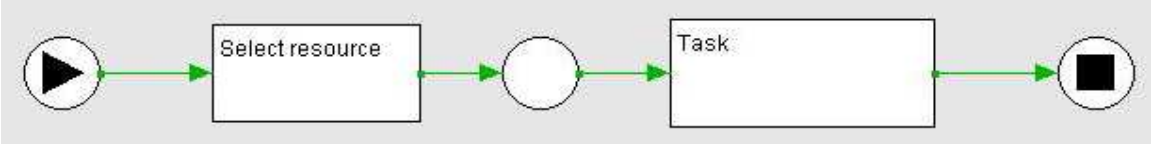


Figure 4-2 Direct allocation workflow resource pattern represented in terms of petri nets

Once we have a process defined in the workflow editor and the representation of the applied patterns, we can transform it to the input specification of the target application. We will show how to do that in the next sections, but now we can see the result in a graphical way (see Figure 4-3).

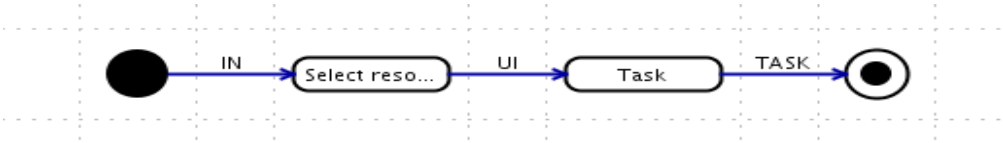


Figure 4-3 Petri nets process transformed into state charts in Atoms

Figure 4-3 shows us the result of the transformation of a task, which has the direct allocation pattern applied, into state charts in Atoms. As we can see in former chapters, petri nets notation uses tasks to define the processes whereas state charts uses states. In this transformation, we will transform each task defined in petri nets into an arrow (or transition) between states. Each state will be called as the task it has to execute later and, the transition

that is called UI (user interface), will have the same name as its source state (in this case, Select resource).

Next section will show us the description of the workflow resource patterns and the representation of its behaviour and user interfaces used to transform them.

4.2 Workflow resource patterns

The Workflow Patterns initiative is a joint effort of Eindhoven University of Technology and Queensland University of Technology which started in 1999. The aim of this initiative is to provide a conceptual basis for process technology. In particular, the research provides a thorough examination of the various perspectives:

1. **Control flow**
2. **Data**
3. **Resource**
4. **Exception handling**

These perspectives need to be supported by a workflow language or a business process modeling language.

In this thesis we will deal with workflow resource patterns, which cover the various ways in which resources are represented and used in workflows. Inside the workflow resource pattern there are four categories:

1. **Creation patterns**
2. **Push patterns**
3. **Pull patterns**
4. **Auto-start patterns**

Each category will be defined, containing the definition of each pattern that it covers in the same way, using a template as follows:

Identifier:	Identifier of the pattern.
Name of the pattern:	Name of the pattern.
Synopsis:	The definition of the pattern.
Context:	When the application of this pattern is useful.
Examples:	It would be a good application of the pattern: Example.
Petri nets representation: Representation of the pattern in terms of petri nets.	

Table 4-1 Template to define workflow patterns

Then, we will summarize about what commercials software and workflow notations (UML, BPMN and BPEL) support the given pattern, and what difficulties we have found to represent these patterns in the target software, Atoms.

RESOURCE PATTERNS	SUPPORTED			IGNORE		ADVERTISE	
CREATION	**Directly**	**Indirectly**	**No support**	**Yes**	**No**	**Yes**	**No**
Direct allocation	X				X		X
Deferred allocation		X			X		X
Authorization based		X			X		X
Separation of duties			X		X	X	
Case handling			X		X	X	
Retain familiar			X		X	X	
Capability-based allocation			X		X	X	
History-based allocation			X		X	X	
Hierarchy level based		X			X		X
PUSH							
Distribution by offer single-resource		X		X			X
Distribution by offer multiple-resources		X		X		X	
Distribution by allocation single-resource	X				X		X
Random allocation			X		X	X	
Round robin allocation			X		X	X	
Shortest queue			X		X	X	
Early distribution			X	X			X
Distribution on enablement	X			X			X
Late distribution			X	X			X
PULL							
Resource-initiated allocation			X	X			X
Resource-initiated execution-allocated task	X			X			X
Resource-initiate execution-offered task			X	X			X
System determined agenda content			X	X			X
Resource determined agenda content	X			X			X
Selection autonomy	X			X			X
DETOUR							
Delegation		X			X		X
Escalation		X			X		X
Deallocation			X		X	X	
Stateful reallocation			X		X	X	
Stateless reallocation		X			X		X
Suspension / resumption			X	X			X
Skip		X			X		X
Redo		X			X		X
Pre-Do			X	X			X
AUTO-START							
Commencement on creation			X	X			X
Commencement on allocation	X			X			X
Piled execution			X	X			X
Chained execution		X		X			X

Table 4-2 Summary of the workflow resource patterns analysis

Table 4-2 shows us all workflow resource patterns, summarizing the conclusions we have obtained after study the target application features. In this table, we can see what patterns are directly, indirectly or not supported by Atoms, what patterns we are going to ignore if we find it applied in a task, and what patterns will need to send an advertisement to the workflow manager.

4.2.1 Creation patterns

Creation Patterns correspond to limitations on the manner in which a work item may be executed. They are specified at design time, usually in relation to a task, and serve to restrict the range of resources that can undertake work items that correspond to the task. They also

influence the manner in which a work item can be matched with a resource that is capable of undertaking it.

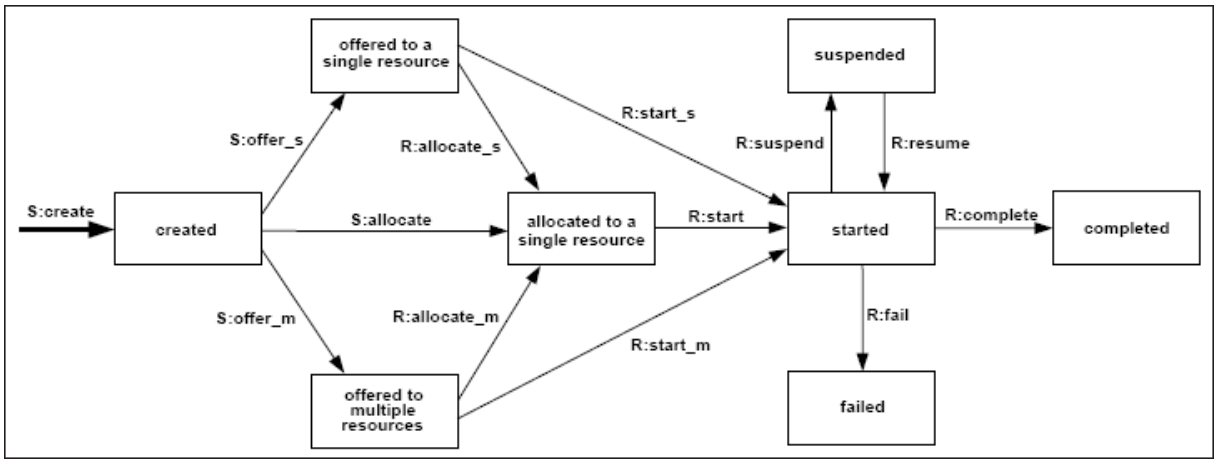


Figure 4-4 Creation patterns

4.2.1.1 Direct allocation

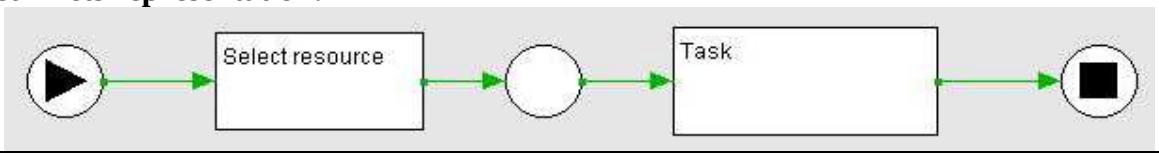
Identifier:	R-DA
Name of the pattern:	Direct allocation
Synopsis:	The ability to specify at design time the identity of the resource that will execute a task. This pattern prevents the problem of non-suitable allocation but there is no opportunity to change the resource if he is not available to perform the task.
Context:	Apply this pattern is a good option when we need to control the resource assignment of a task.
Examples:	It would be a good application of the pattern: "Upload new web contents" task must only be undertaken by Administrator.
Petri nets representation:	

Table 4-3 Direct allocation analysis

Direct allocation, the simplest pattern, is able to apply in any case. There is neither commercial software nor notation which does not support this pattern.

4.2.1.2 Deferred allocation

Identifier:	R-FBA
Name of the pattern:	Deferred allocation
Synopsis:	The ability to defer specifying the identity of the resource that will execute a task until runtime.
Context:	Apply this pattern is a good option when the resource identity has to be changed dynamically during the workflow execution to ensure that the

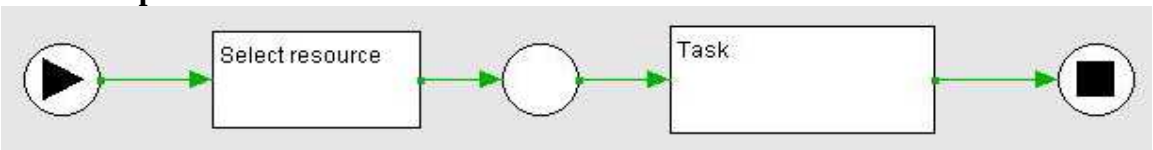
	resource is the most appropriate to develop the task.
Examples:	It would be a good application of the pattern: "Pay a payment order" task will be executed only by the resource named in the next-resource field of the payment order. It would not be a good application of the pattern: "Prepare the order" in the warehouse can not defer the information of any warehouseman directly from the order. The order only contains information about the customer.
Petri nets representation:	

Table 4-4 Deferred allocation analysis

In case of commercial software, some of the most popular ones like Staffware or Websphere MQ support it directly, whereas the notations BPMN and UML do not support that.

In this case, we can not know what the resource that has to execute the task is. Then, the representation will be the same as direct allocation, choosing the resource in a list directly.

4.2.1.3 Authorization based

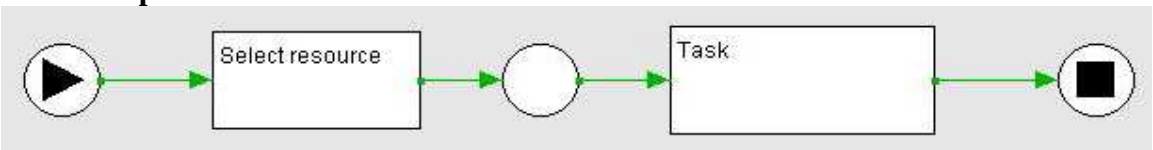
Identifier:	R-RA
Name of the pattern:	Authorization
Synopsis:	The ability to specify the range of resources that are authorized to execute a task.
Context:	Apply this pattern is a good option when it is necessary to define a security framework over a workflow implementation that is independent of the way in which tasks are actually routed in runtime.
Examples:	It would be a good application of the pattern: Only the "Social worker" is authorized to execute instances of the "Apply the Final Interview" task.
Petri nets representation:	

Table 4-5 Authorization pattern analysis

Neither Staffware nor Websphere MQ support this pattern. In terms of notation, neither BPMN nor UML nor BPEL support it.

IMmedia Atoms does not support it too. In this case, the authorization will be in charge of the workflow manager, who will have to do a direct allocation to the task defined with this pattern taking in the account that not all the resources are able to execute it.

4.2.1.4 Separation of duties

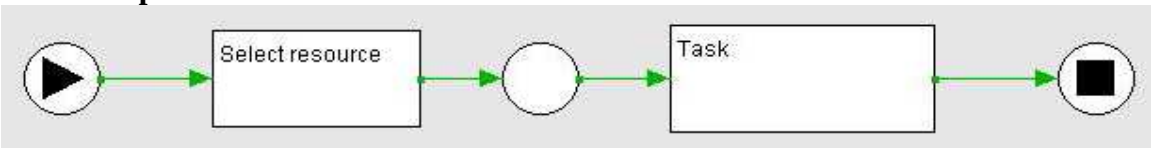
Identifier:	R-SOD
Name of the pattern:	Separation of duties
Synopsis:	The ability to specify that two tasks must be allocated to different resources in a given process.
Context:	Apply this pattern is a good option when two tasks can not be executed by the same resource to assure the intervention of different resources.
Examples:	It would be a good application of the pattern: "Second medical opinion" can not be executed by the same doctor who executed the past task "Medical diagnostic" It would not be a good application of the pattern: "Revise the order" and "Approve the order" are two tasks that have to be executed by the same resource.
Petri nets representation:	

Table 4-6 Separation of duties pattern analysis

While Staffware does not support this pattern, Websphere MQ supports it directly via task linking between activities in the process model that can not have the same resource allocation at runtime within a case. In terms of workflow notations, they do not allow specifying the separation of duties in terms of relationships between tasks, nor they allow the separation of duties based on security mechanisms. Thus this pattern is not supported neither by BPMN, UML nor BPEL.

In this case, we can only advertise the workflow manager that a Separation of duties pattern has been defined and he will have to manage it manually.

4.2.1.5 Case handling

Identifier:	R-CH
Name of the pattern:	Case handling
Synopsis:	The ability to allocate the tasks within a given process to the same resource.
Context:	Apply this pattern is a good option when a process needs to be executed entirely by the same resource.
Examples:	It would be a good application of the pattern: All tasks in "Revise the daily work" have to be executed by the supervisor. It would not be a good application of the pattern: "Second medical opinion" can not be executed by the same doctor who executed the past task "Medical diagnostic".
Petri nets representation:	

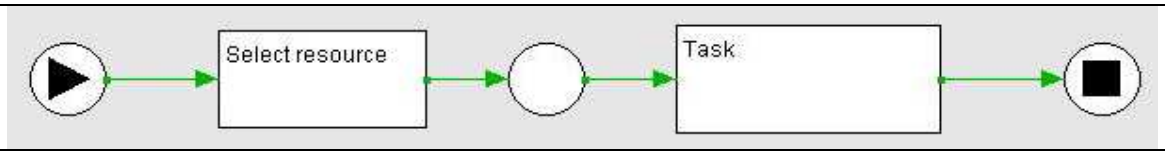


Table 4-7 Case handling pattern analysis

Neither Staffware nor Websphere MQ support this pattern and, whereas BPMN and UML neither support it, BPEL has a feature of dynamic assignment using an expression that allows specifying that a next task must be assigned to the resource who executed the previous task. In this case, BPEL is the only notation that supports this pattern (excluding YAWL, of course).

In this case, we can only advertise the workflow manager that a Case handling pattern has been defined and he will have to manage it manually.

4.2.1.6 Retain familiar

Identifier:	R-RF
Name of the pattern:	Retain Familiar
Synopsis:	Where several resources are available to undertake a task, the ability to allocate a task within a given process to the same resource that undertook a preceding task.
Context:	Apply this pattern is a good option when a sequence of tasks need to be executed entirely by the same resource.
Examples:	It would be a good application of the pattern: The tasks “Revise the stock” and “Make a production request” have to be executed by the same resource. It would not be a good application of the pattern: “Second medical opinion” can not be executed by the same doctor who executed the past task “Medical diagnostic”.
Petri nets representation:	

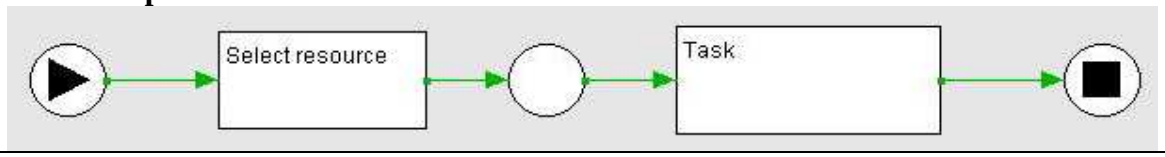


Table 4-8 Retain familiar pattern analysis

Websphere MQ supports it indirectly. It can allocate a common resource, which can be specified for specific tasks in the process model requiring the same resource allocation at runtime within a case. Staffware does not support it.
In terms of notations, neither BPMN nor UML support it whereas BPEL has a function that lets the workflow manager to get the previous task approver.

In this case, and like the last two patterns, we can only advertise the workflow manager that a Retain familiar pattern has been defined and he will have to manage it manually.

4.2.1.7 Capability-based allocation

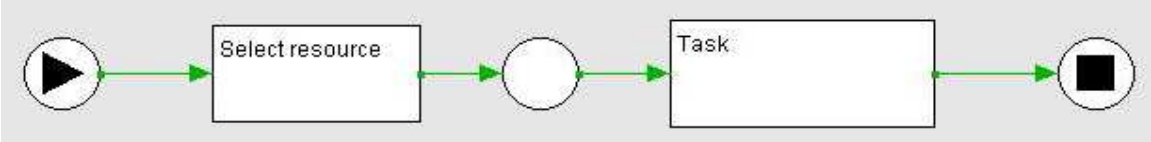
Identifier:	R-CBA
Name of the pattern:	Capability-based allocation
Synopsis:	The ability to offer or allocate instances of a task to resources based on specific capabilities that they possess.
Context:	Apply this pattern is a good option when it is necessary an expert to execute a task.
Examples:	It would be a good application of the pattern: Instances of the "Airframe Examination" task should be allocated to an Engineer with an aeronautics degree, an Airbus in-service accreditation and more than 10 years experience in Airbus servicing. It would not be a good application of the pattern: The task "Recollect goods of an order" should not need an expert to be executed.
Petri nets representation:	

Table 4-9 Capability-based allocation pattern analysis

Neither Staffware nor Websphere MQ and neither BPMN nor UML support this pattern. In the other hand, BPEL allows defining user properties and store them in a file, which become accessible via some functions.

It is not possible to do in Atoms. The workflow manager will have to select a suitable resource to execute the task.

4.2.1.8 History-based allocation

Identifier:	R-HBA
Name of the pattern:	History-based allocation
Synopsis:	The ability to offer or allocate tasks to resources on the basis of their previous execution history.
Context:	Apply this pattern is a good option when a task needs to be executed by a resource that has executed it before.
Examples:	It would be a good application of the pattern: Allocate the "Finalize heart bypass" task to the surgeon who has successfully completed the most of these tasks. It would not be a good application of the pattern: Allocate the "Prepare the order" in a warehouse, where there are 20 warehouseman, always to the same worker would not be useful.
Petri nets representation:	

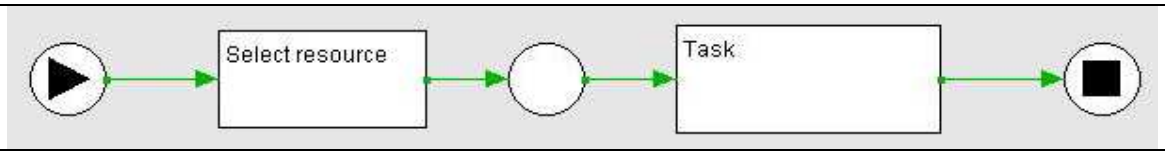


Table 4-10 History-based allocation pattern analysis

Neither Staffware nor Websphere MQ support this pattern. The same for BPMN, UML and BPEL. However, BPEL allows implementing this feature, so using this notation could be possible to use this pattern.

It is not possible to represent, there is no history information stored in Atoms. Then, the workflow manager has to know which resources have experience in the execution of this task and select a suitable one.

4.2.1.9 Hierarchy level based

Identifier:	R-HLB
Name of the pattern:	Hierarchy level based
Synopsis:	The ability to offer or allocate instances of a task to resources based on their hierarchic level within the organization and/or their relationship with other resources.
Context:	Apply this pattern is a good option when a task needs to be executed by a resource that has a specific range in a company.
Examples:	It would be a good application of the pattern: The task "Reduce the cost of the project" has to be executed by a "Financial Manager" who has a high position in his company.
Petri nets representation:	

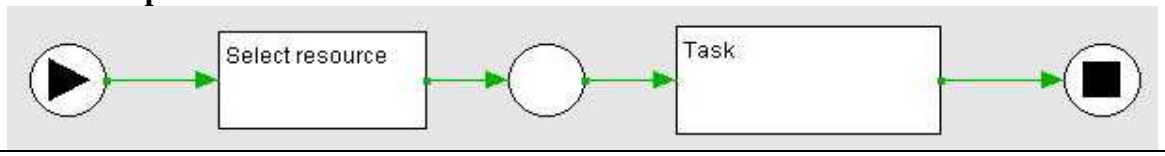


Table 4-11 Hierarchy level based pattern analysis

Staffware supports partially roles and groups, which can be used to define some hierarchies. In case of Websphere MQ, it offers a directly support of this pattern. Neither BPMN nor UML support it, whereas BPEL stores the organizational structure and it can be accessed.

In case of Atoms, there is no organizational structure stored and the allocations are always directly done. Then, the best option is to define groups of users and select one of them before the execution of the task.

4.2.2 Push patterns

Push Patterns characterize situations where newly created work items are proactively offered or allocated to resources by the system. These may occur indirectly by advertising work items to selected resources via a shared work list or directly with work items being allocated to specific resources. In both situations however, it is the system that takes the initiative and causes the distribution process to occur.

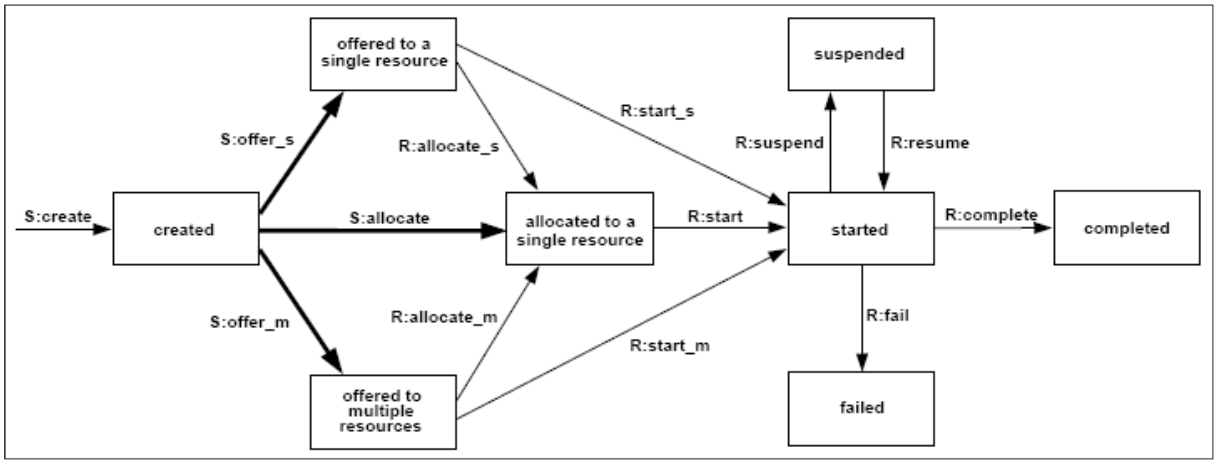


Figure 4-5 Push patterns

4.2.2.1 Distribution by offer single-resource

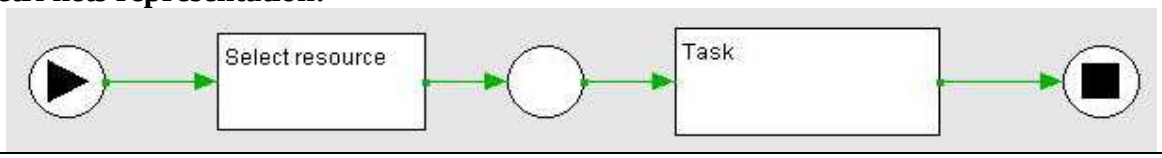
Identifier:	R-DBOS
Name of the pattern:	Distribution by offer single-resource
Synopsis:	The ability to offer a task to a selected individual resource.
Context:	Apply this pattern is a good option when we know who has to execute a given task.
Examples:	It would be a good application of the pattern: "Upload new web contents" task is assigned to the Administrator.
Petri nets representation:	

Table 4-12 Distribution by offer single resource pattern analysis

Neither Staffware nor Websphere MQ support this pattern. The same for BPMN and UML. However, BPEL allows offering a work item to members of a group and a user can "acquire" the offered work item.

In Atoms, the tasks in process are allocated directly to the resource assigned (or to a group of resources) who can execute the task when it wants. Then, this pattern is supported by Atoms indirectly.

4.2.2.2 Distribution by offer multiple-resources

Identifier:	R-DBOM
Name of the pattern:	Distribution by offer multiple-resources
Synopsis:	The ability to offer a task to a group of selected resources.
Context:	Apply this pattern is a good option when we have to allocate a task to a group of workers or a specific department.
Examples:	It would be a good application of the pattern: "Approve the buying request" has to be executed by any worker in the

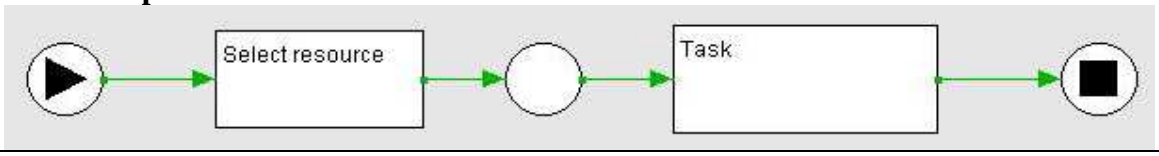
	"Commercial department" It would not be a good application of the pattern: "Accept company merge" has to be executed by the "General Director", not by "Direction department" in general.
Petri nets representation: 	

Table 4-13 Distribution by offer multiple resource pattern analysis

As Staffware as Websphere MQ support this pattern using group queues. Whereas BPMN and UML do not support it, BPEL can specify the name of a group as an assignee of the task. As a result, the task will be offered to all members of a group.

As the last pattern explained, we can select a group of resources and allocate a task to all of its members. Then, any resource inside the group can execute it.

4.2.2.3 Distribution by allocation single-resource

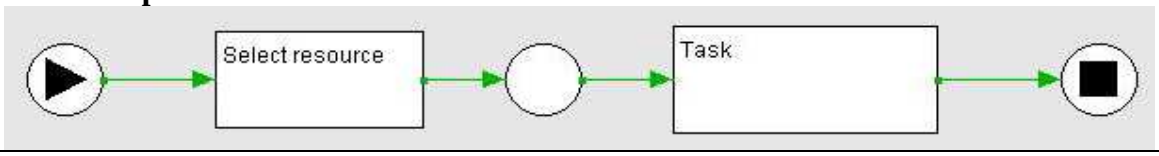
Identifier:	R-DBAS
Name of the pattern:	Distribution by allocation single-resource
Synopsis:	The ability to directly allocate a task to a specific resource for execution.
Context:	Apply this pattern is a good option when we realize that a resource is very busy and we decide to change it.
Examples:	It would be a good application of the pattern: "Develop the final test" task should be allocated to the "Chemical engineer".
Petri nets representation: 	

Table 4-14 Distribution by allocation single resource pattern analysis

Another one of the simplest patterns can be supported by the commercial software and the business process notations.

In Atoms, this is the standard behaviour of any allocation. It offers direct support for this pattern.

4.2.2.4 Random allocation

Identifier:	R-RMA
Name of the pattern:	Random allocation
Synopsis:	The ability to offer or allocate tasks to suitable resources on a random basis.

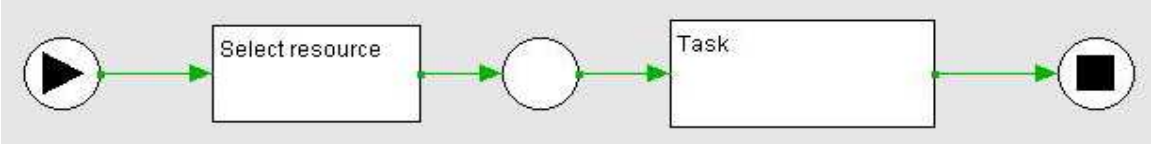
Context:	Apply this pattern is a good option when it does not matter which resource execute a given task.
Examples:	It would be a good application of the pattern: The "Judge case" work item is allocated to a Magistrate on a random basis. It would not be a good application of the pattern: "Accept company merge" can not be executed by any worker of the company.
Petri nets representation:	

Table 4-15 Random allocation pattern analysis

Neither Staffware nor Websphere MQ support it. BPMN and UML neither. Only BPEL can use some features that allow the simulation of it indirectly.

There are no features comparable to this way of allocation. Then, the workflow manager will be in charge of select the resource.

4.2.2.5 Round robin allocation

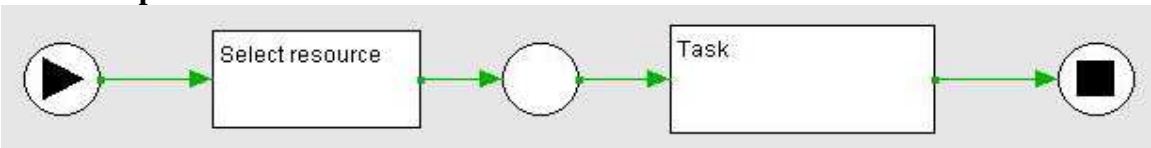
Identifier:	R-RRA
Name of the pattern:	Round robin allocation
Synopsis:	The ability to allocate a task to available resources on a cyclic basis.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: Work items corresponding to the "Umpire Match" task are allocated to each available Referee on a cyclic basis. It would not be a good application of the pattern: "Revise the daily work" has to be executed always by the supervisor, not by a group of users in a cyclic basis.
Petri nets representation:	

Table 4-16 Round robin allocation pattern analysis

Neither Staffware nor Websphere MQ support it. Neither BPMN nor UML too. Only BPEL can use some features that allow the simulation of it indirectly like the random allocation.

As a variation of the last pattern, Atoms does not offer support to it and we propose the same solution.

4.2.2.6 Shortest queue

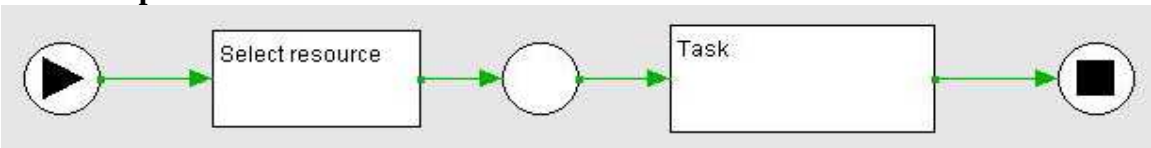
Identifier:	R-SHQ
Name of the pattern:	Shortest queue
Synopsis:	The ability to allocate a task to the resource that has the least number of tasks allocated to it.
Context:	Apply this pattern is a good option when we need the most available resource, the one with less tasks allocated, to execute a given task.
Examples:	It would be a good application of the pattern: "Show new apartment" task is allocated to the "Sales representative" who has the least number of tasks to do. It would not be a good application of the pattern: "Revise the daily work" can not be executed by the most available resource, it has to be executed by the "Supervisor" always.
Petri nets representation:	

Table 4-17 Shortest queue pattern analysis

Neither Staffware nor Websphere MQ support it. In terms of notations, BPMN and UML do not support it whereas BPEL can support it indirectly implementing a personalized service.

There are no features in Atoms that allow the workflow manager to do an allocation like that. The workflow manager will have to know how busy are the workers and allocate this task manually.

4.2.2.7 Early distribution

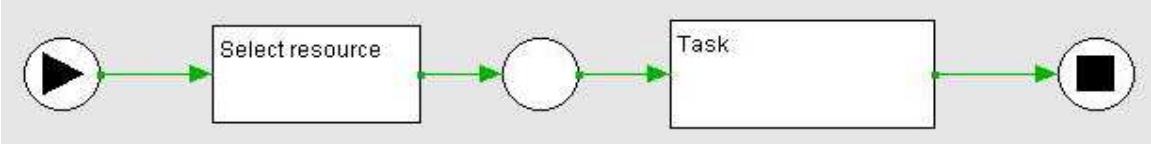
Identifier:	R-ED
Name of the pattern:	Early distribution
Synopsis:	The ability to advertise and potentially allocate tasks to resources ahead of the moment at which the tasks are enabled.
Context:	Apply this pattern is a good option when we need to inform the resource about it has to execute a task before its execution.
Examples:	It would be a good application of the pattern: "Organize the annual college reunion" task is allocated to the "Secretary" of ex-students department at least three months prior to the time that it will commence.
Petri nets representation:	

Table 4-18 Early distribution pattern analysis

Neither Staffware nor Websphere MQ support this pattern. Any notation supports it neither.

Atoms does not offer support for this pattern neither. We will ignore this pattern in the definition, being automatically converted to Distribution on enablement pattern, which corresponds to the standard behaviour of the system.

4.2.2.8 Distribution on enablement

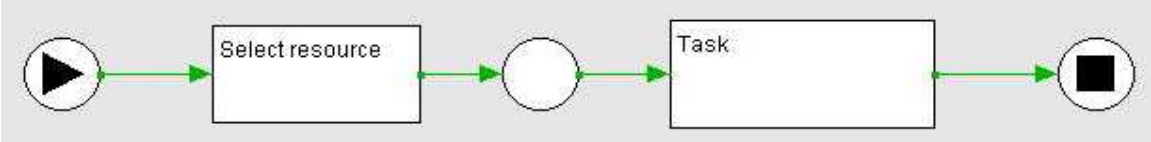
Identifier:	R-DE
Name of the pattern:	Distribution on enablement
Synopsis:	The ability to advertise and allocate tasks to resources at the moment they are enabled for execution.
Context:	Apply this pattern is a good option when don't need to advertise the resource of a task prior to its execution.
Examples:	It would be a good application of the pattern: "Prepare the payroll" task is allocated to the "Accountant assistant" at the time it is required to execute It would not be a good application of the pattern:
Petri nets representation:	

Table 4-19 Distribution on enablement pattern analysis

This pattern represents the standard behaviour in work item distribution in the commercial software and in the business process notations.

Atoms has this behaviour in its distribution. We do not have to do anything to accomplish this pattern.

4.2.2.9 Late distribution

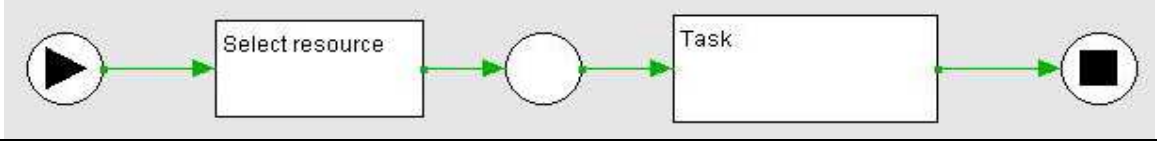
Identifier:	R-LD
Name of the pattern:	Late distribution
Synopsis:	The ability to advertise and allocate tasks to resources after the task has been enabled.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: "Pack milk boxes" task is allocated to a Packer after they have been sealed to be delivered.
Petri nets representation:	

Table 4-20 Late distribution pattern analysis

Like the early distribution pattern, neither Staffware nor Websphere MQ and any notation support this pattern.

We will act like in the Early distribution pattern, ignore it and will be automatically converted to Distribution on enablement pattern.

4.2.3 Pull patterns

Pull Patterns correspond to the situation where individual resources are made aware of specific work items, that require execution, either via a direct offer from the system or indirectly through a shared work list. The commitment to undertake a specific task is initiated by the resource itself rather than the system. Generally this results in the work item being placed on the specific work list for the individual resource for later execution although in some cases, the resource may elect to commence execution on the work item immediately.

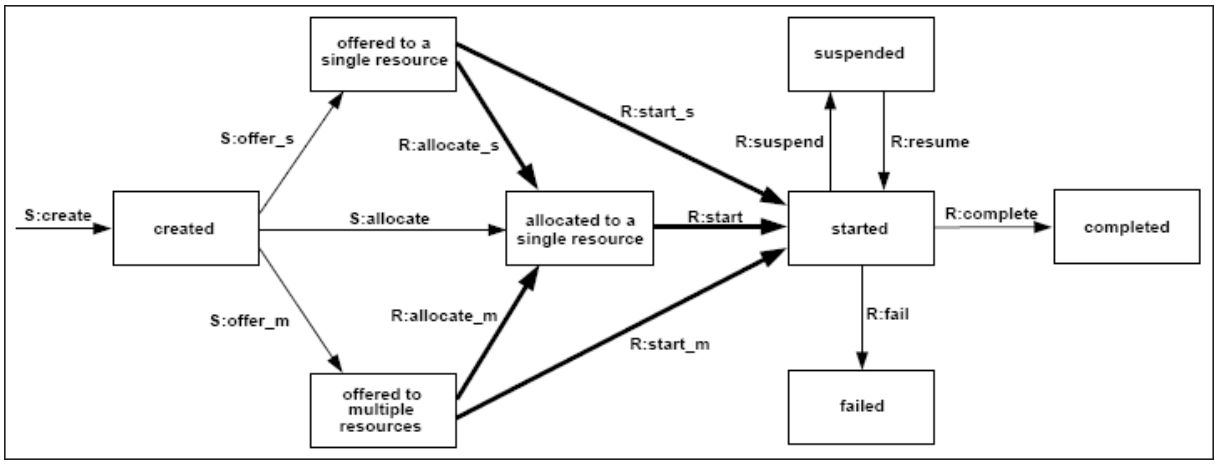


Figure 4-6 Pull patterns

4.2.3.1 Resource-initiated allocation

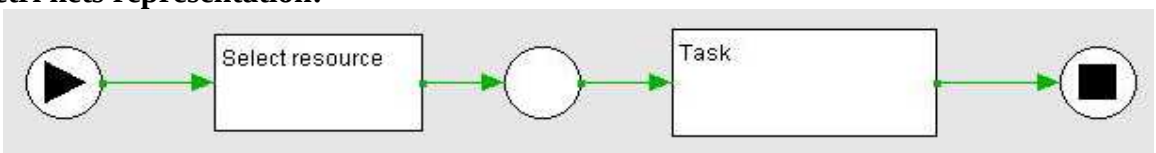
Identifier:	R-RIA
Name of the pattern:	Resource-initiated allocation
Synopsis:	The ability for a resource to commit to undertake a task without needing to commence working on it immediately.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The “commercial” accept a task he has offered in his agenda to execute it in the future.
Petri nets representation:	

Table 4-21 Resource-initiated allocation pattern analysis

Neither Staffware nor Websphere MQ support it. In terms of notation, neither BPMN nor UML nor BPEL support it.

Atoms does not offer support to this pattern neither. It is not necessary to do anything, the task will be allocated without asking for permission to the resource.

4.2.3.2 Resource-initiated execution-allocated task

Identifier:	R-RIEA
Name of the pattern:	Resource-initiated execution-allocated task
Synopsis:	The ability for a resource to start to work on a task that is allocated to it.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Laser Printer selects the next Printing document task which is allocated to it and commences work on it.
Petri nets representation:	

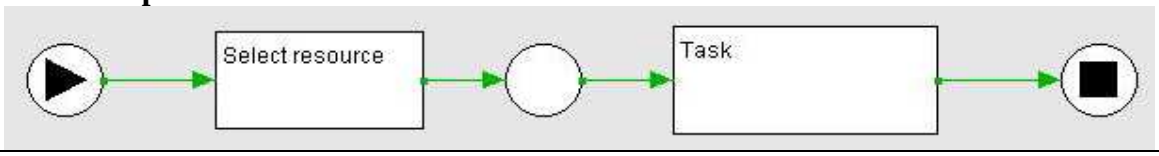


Table 4-22 Resource-initiated execution-allocated task pattern analysis

This is the standard consequence of starting an item on a work queue in Staffware and Websphere MQ. Notations like BPMN and UML do not support this pattern, but BPEL supports it directly.

Atoms runs like Staffware and Websphere MQ, so its standard behaviour involves the support for this pattern.

4.2.3.3 Resource-initiated execution-offered task

Identifier:	R-RIEO
Name of the pattern:	Resource-initiated execution-offered task
Synopsis:	The ability for a resource to select a task offered to it and starts to work on it immediately.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Courier Driver selects the next “Delivery” task from those offered and commences work on it.
Petri nets representation:	

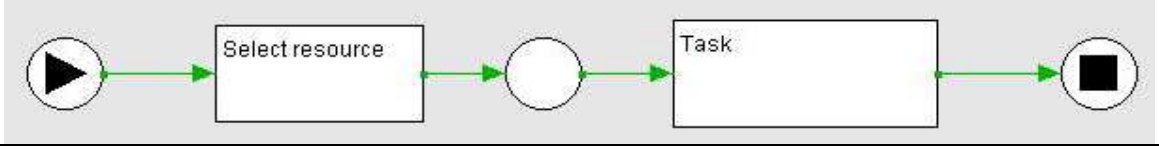


Table 4-23 Resource-initiated execution-offered task pattern analysis

This approach to work distribution is adopted by Staffware whereas WebSphere MQ using work queues, where the work items wait for an acceptation of a resource to execute it. BPEL adopt a similar approach of that whereas BPMN and UML do not support it.

In Atoms it is not possible to offer a task, so this pattern is not supported.

4.2.3.4 System determined agenda content

Identifier:	R-SDAC
Name of the pattern:	System determined agenda content
Synopsis:	The ability of the work flow engine to order the content and sequence in which tasks are presented to a resource for execution.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: Depending on the configuration specified in the process model, the workflow engine presents work items to resources either in order of work item priority or date created.
Petri nets representation:	

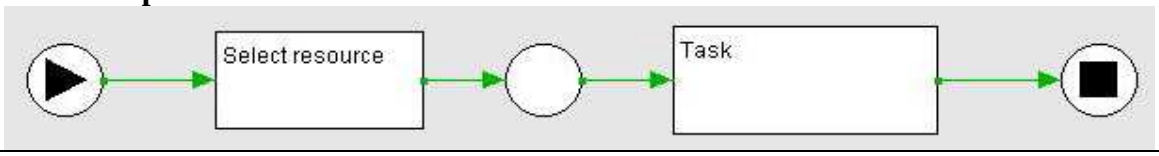


Table 4-24 System determined agenda content pattern analysis

Staffware let ordering work items by priority and Websphere MQ does not support it. In terms of notations, neither BPMN nor UML nor BPEL support it.

Atoms allows the user to order his work list.

4.2.3.5 Resource determined agenda content

Identifier:	R-RDAC
Name of the pattern:	Resource determined agenda content
Synopsis:	The ability for resources to specify the format and content of tasks listed in the agenda for execution.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Printer has a work list ordered by time of arrival.
Petri nets representation:	

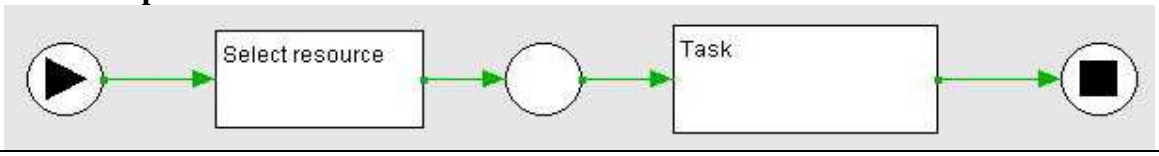


Table 4-25 Resource determined agenda content pattern analysis

Staffware, WebSphere MQ and Oracle BPEL allow any work item attribute to be used as the basis of the sort criterion or for filtering the work items that are displayed, whereas BPMN and UML do not support it.

Atoms allows the user to order his work list.

4.2.3.6 Selection autonomy

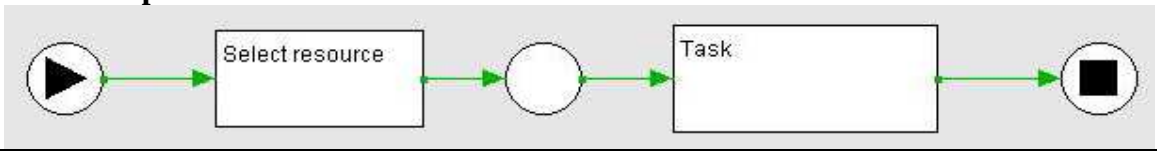
Identifier:	R-SA
Name of the pattern:	Selection autonomy
Synopsis:	The ability for resources to select a task for execution based on task attributes, features and capabilities of the resource.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: Of the outstanding "Pruning" work items, the Head Gardener chooses the one for execution they feel they are best suited to.
Petri nets representation:	

Table 4-26 Selection autonomy pattern analysis

All of the workflow engines examined, except BPMN and UML which do not support any notion of agenda handler, provide support for this pattern.

In Atoms, a resource can choose what task wants to execute without any restriction, only having allocated to it.

4.2.4 Detour patterns

Detour Patterns refer to situations where work item distributions that have been made for resources are interrupted either by the system or at the instigation of the resource. As a consequence of this event, the normal sequence of state transitions for a work item is varied.

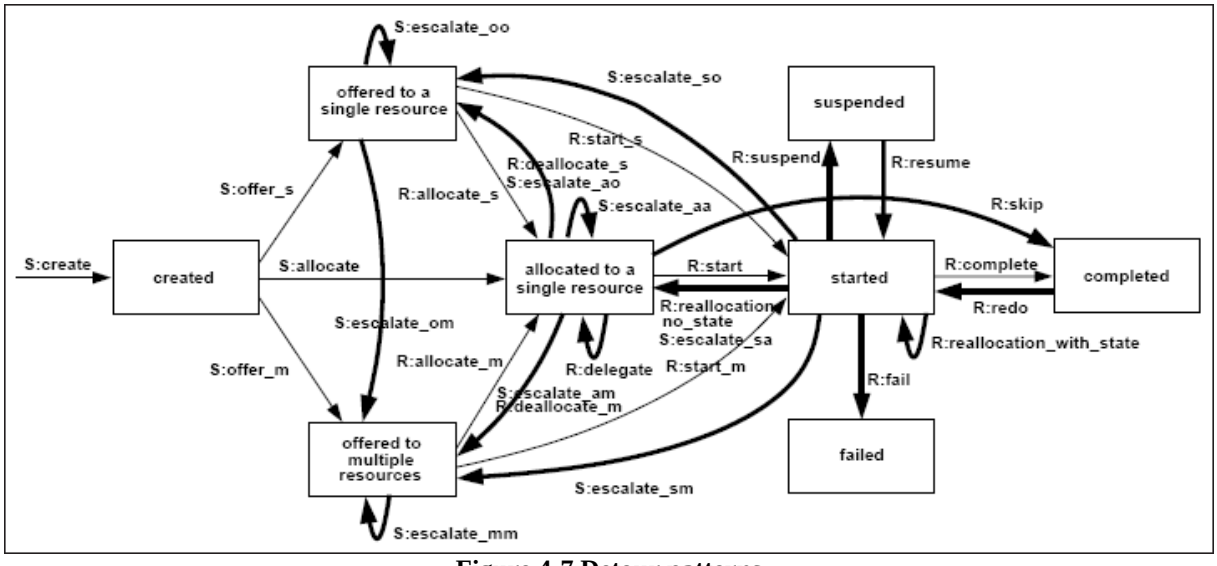


Figure 4-7 Detour patterns

4.2.4.1 Delegation

Identifier:	R-D
Name of the	Delegation

pattern:	
Synopsis:	The ability for a resource to allocate a task previously allocated to it to another resource.
Context:	Apply this pattern is a good option when a resource can not execute a task and decides to reallocate to another resource.
Examples:	It would be a good application of the pattern: The Chief Accountant passed all of their outstanding tasks onto the Assistant Accountant.
Petri nets representation:	

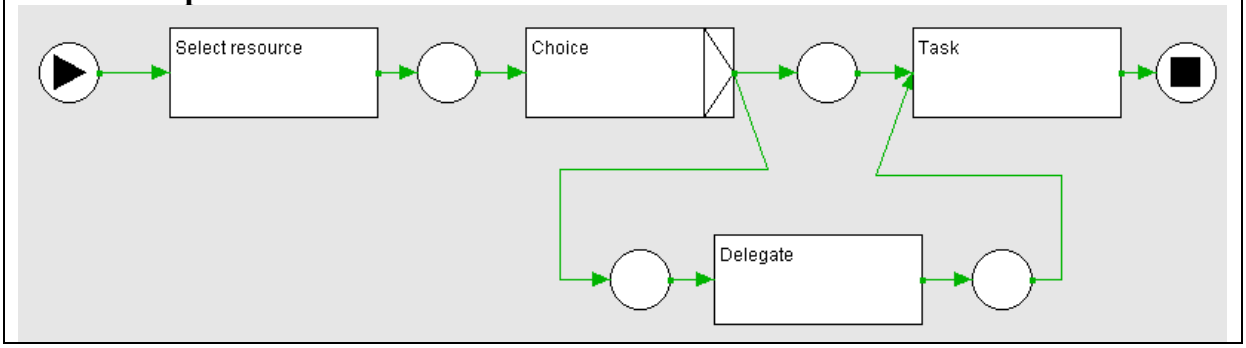


Table 4-27 Delegation pattern analysis

Both Staffware and WebSphere MQ allow a resource to forward any work item that is routed to them to another resource. Whereas BPMN and UML do not support this pattern, BPEL supports it directly by means of reassign actions.

Atoms does not offer support for this pattern. Any kind of change of resource has to be implemented in the process definition, like in our petri nets representation.

4.2.4.2 Escalation

Identifier:	R-E
Name of the pattern:	Escalation
Synopsis:	The ability of the workflow system to offer or allocate a task to a resource or a group of resources other than those it has previously been offered or allocated to in an attempt to expedite the completion of the task.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The “Review earnings” task was reallocated to the Financial Center Office. It had previously been allocated to the Financial Accountant but the deadline for completion had been exceeded.
Petri nets representation:	

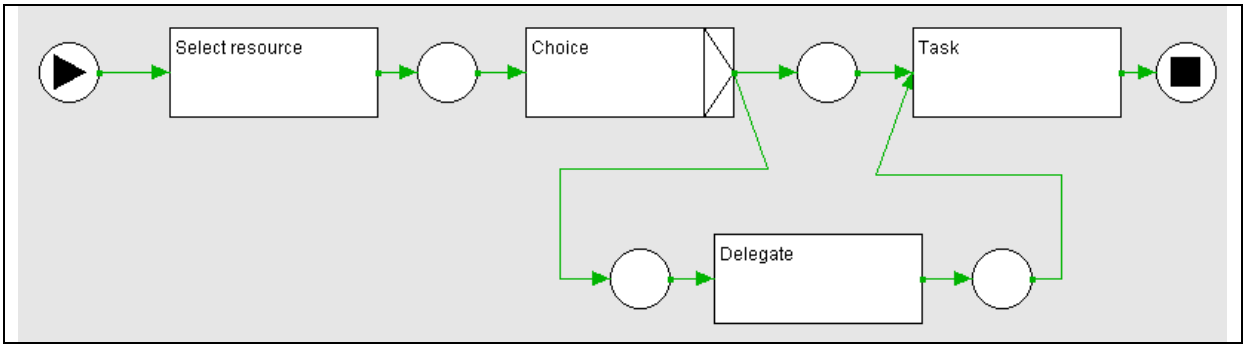


Table 4-28 Escalation pattern analysis

Like the last pattern, Staffware, Websphere MQ and BPEL notation support the reallocation of a task whereas BPMN and UML not.

We will tackle this pattern as we have done with the last one. In fact, it is the same behaviour in different situations.

4.2.4.3 Deallocation

Identifier:	R-SD
Name of the pattern:	Deallocation
Synopsis:	The ability to a resource to relinquish a task which is allocated to it and make it available for allocation to another resource.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Supervisor has a lot of tasks allocated and decides to allocate one of his tasks to a subordinate that will help him.
Petri nets representation:	

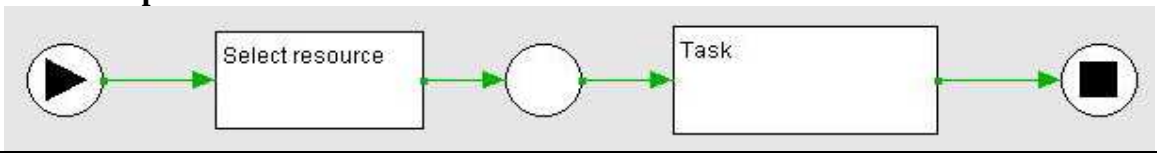


Table 4-29 Deallocation pattern analysis

Neither Staffware nor Websphere MQ support it due to the lack of mechanisms to have a task without allocation. The same occurs using BPMN and UML notations, whereas using BPEL, a resource of a group of resource can acquire a work item allocated to its group, and after it can relinquish it, returning the allocation to its group.

Atoms can not leave a task without allocation. For that reason, we will tackle this pattern as we have done the Delegation and the Escallation.

4.2.4.4 Stateful reallocation

Identifier:	R-PR
Name of the pattern:	Stateful reallocation
Synopsis:	The ability of a resource to allocate a task to another resource without loss of state data.

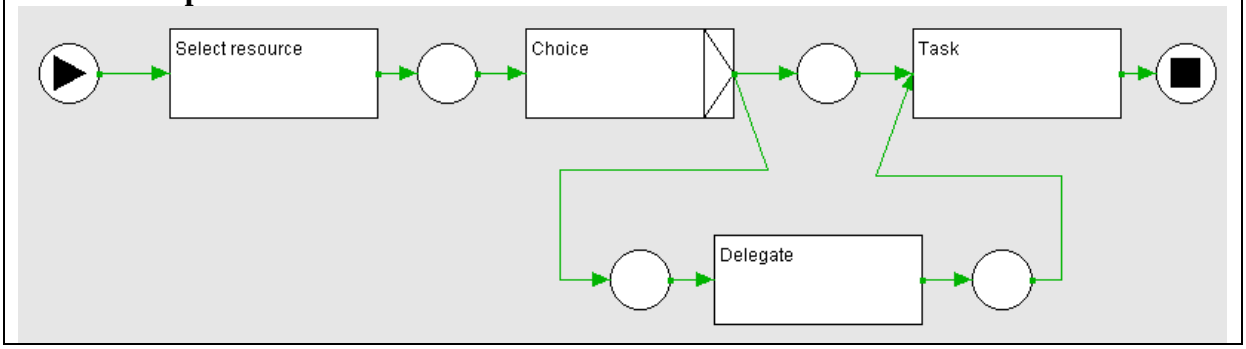
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Senior Partner has suspended work on the "Building Society Audit Plan" task and passed it to the Junior Project Manager for further work. It would not be a good application of the pattern:
Petri nets representation:	

Table 4-30 Stateful reallocation pattern analysis

As we have commented before, Staffware and Websphere MQ allows the reallocation of tasks. BPMN and UML do not support it whereas BPEL allows the reassignment of it work items.

Atoms can not allow the resources to reallocate a task during its execution, so the reallocation has to be done before starting to execute. We will transform it as we have done with the last patterns.

4.2.4.5 Stateless reallocation

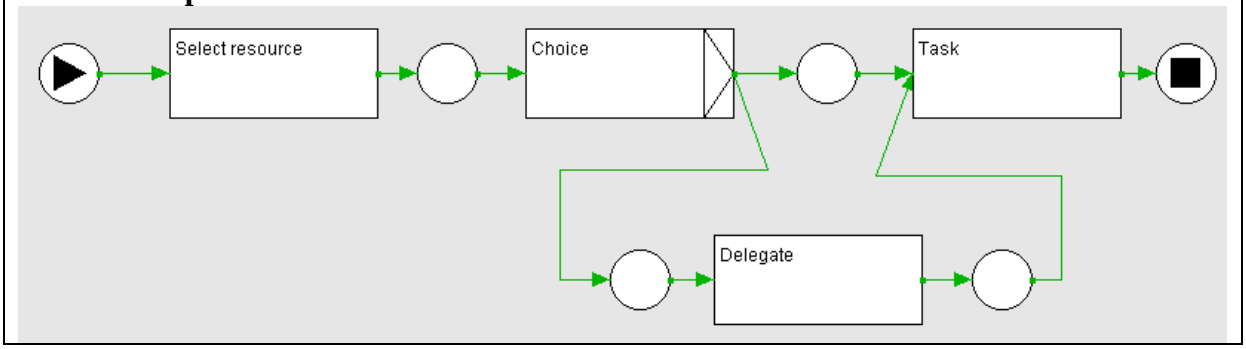
Identifier:	R-UR
Name of the pattern:	Stateless reallocation
Synopsis:	The ability for a resource to reallocate a task currently being executed to another resource without retention of state.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The "Fill form" task has been reallocated to another Bank executive who will restart it.
Petri nets representation:	

Table 4-31 Stateless reallocation pattern analysis

Neither any commercial software nor notations offer support for his pattern.

It is the same problem as the last pattern, we will tackle it in the same way.

4.2.4.6 Suspension / resumption

Identifier:	R-SR
Name of the pattern:	Suspension or resumption
Synopsis:	The ability for a resource to suspend and resume execution of a task.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Secretary has suspended all “Board Meeting” tasks while the Board is being reconstituted.
Petri nets representation:	

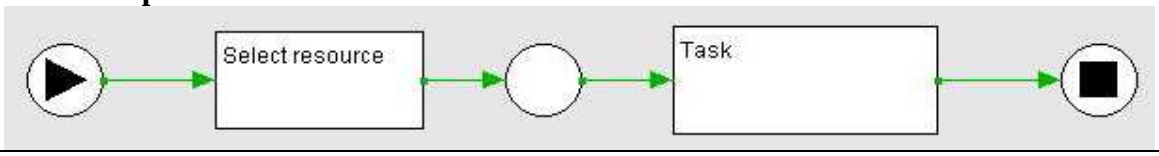


Table 4-32 Suspension/resumption pattern analysis

Staffware allows work items that utilise a form to be suspended at any stage via some options and Websphere MQ does not allow individual work items to be suspended but supports the suspension of an entire workflow case. Neither BPMN nor UML support it, whereas BPEL has available actions that offer support to it.

In terms of Atoms, it is not possible to suspend a task in process. A work item assigned to a resource will wait until the resource executes it.

4.2.4.7 Skip

Identifier:	R-SK
Name of the pattern:	Skip
Synopsis:	The ability for a resource to skip a task allocated to it and to mark the task with a finished status.
Context:	Apply this pattern is a good option when a task can be done automatically from the outside of the process and at the moment of its execution it has been realized before.
Examples:	It would be a good application of the pattern: The Bank teller has elected to skip the “Identify client” task because he considers to the client as Regular client.
Petri nets representation:	

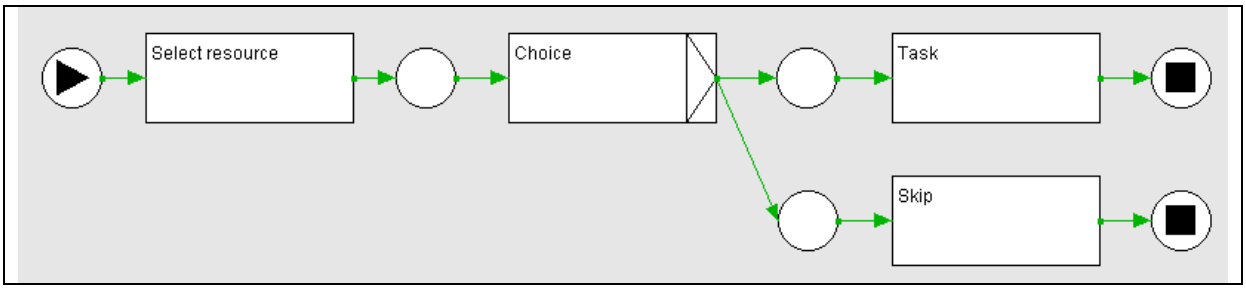


Table 4-33 Skip pattern analysis

Whereas this pattern is not supported by Staffware, Websphere MQ offers directly support to it. Neither BPMN nor UML support it, whereas BPEL supports it directly.

Atoms lets us to define an unconditional task alternatively to another task to skip it, but we will not mark it as done. This is not a problem in Atoms because it does not need to mark all tasks as done, only follow the path defined in the process.

4.2.4.8 Redo

Identifier:	R-RD
Name of the pattern:	Redo
Synopsis:	The ability for a resource to redo a task that has previously been completed in a case.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Inspector has decided to redo the “Interview Key Witness” task.
Petri nets representation:	

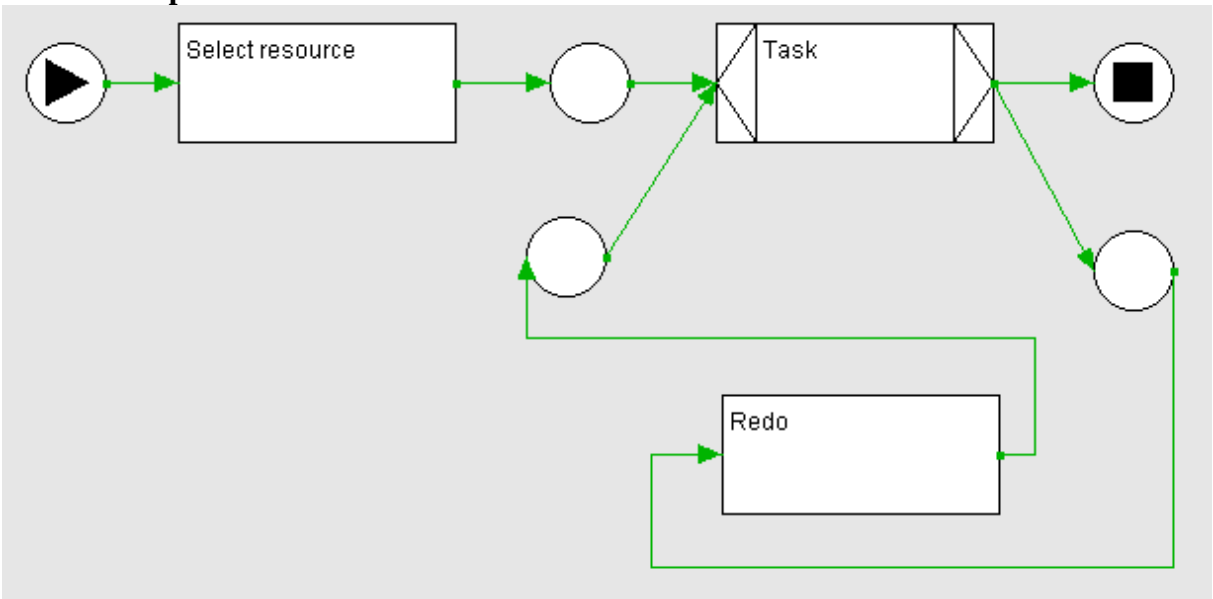


Table 4-34 Redo pattern analysis

Neither the commercial software studied nor the notations offer support for this pattern.

Atoms lets us to define an unconditional transition between two tasks like in the last pattern. Doing that, we can go back and re-do a task already done.

4.2.4.9 Pre-Do

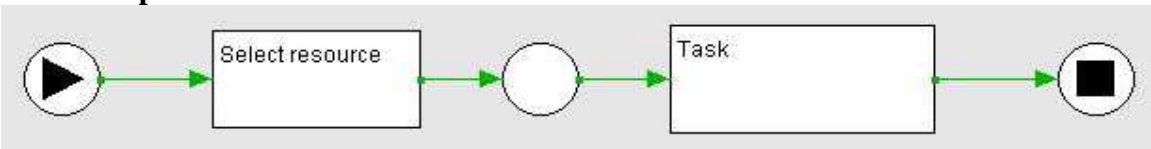
Identifier:	R-PRE
Name of the pattern:	Pre-do
Synopsis:	The ability for a resource to execute a work item ahead of the time that it has been offered or allocated to resources working on a given case.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The Inspector has completed the "Charge Suspect" work item even though the preceding "Interview Witness" work items have not yet been completed.
Petri nets representation:	

Table 4-35 Pre-do pattern analysis

Neither the commercial software studied nor the notations offer support for this pattern.

Atoms does not offer support for this pattern. We will ignore this pattern definition in the transformation and the resource can execute a task when it is available.

4.2.5 Auto-start patterns

Auto-start patterns relate to situations where execution of work items is triggered by specific events in the lifecycle of the work item or the related process definition. Such events may include the creation or allocation of the work item, completion of another instance of the same work item or a work item that immediately precedes the one in question.

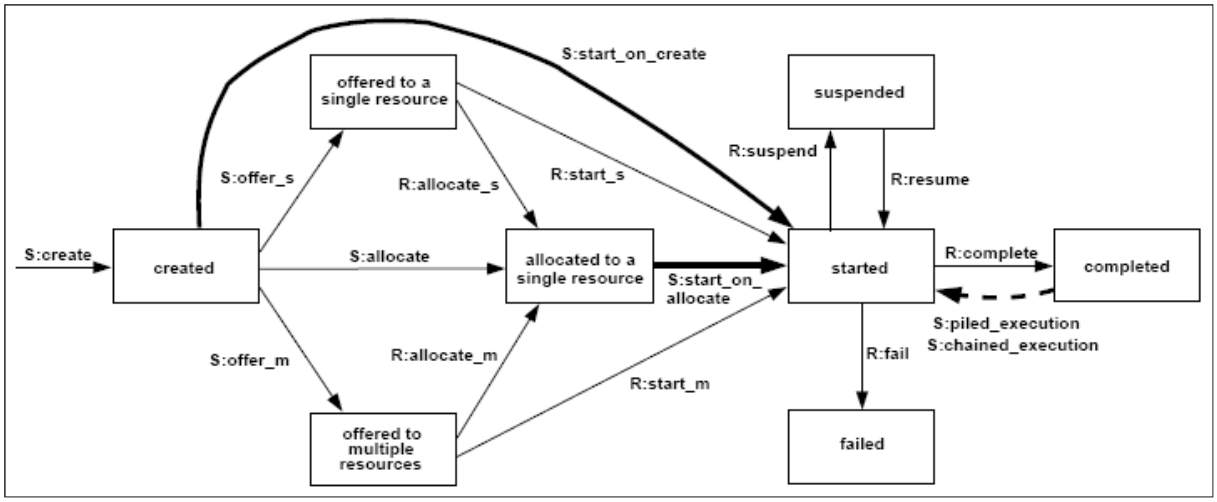


Figure 4-8 Auto-start patterns

4.2.5.1 Commencement on creation

Identifier:	R-CC
Name of the pattern:	Commencement on creation
Synopsis:	The ability for a resource to commence execution on a task as soon as it is created.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The "End of Month" task is allocated to the Chief Accountant who must commence working on it as soon as it is allocated to his work queue.
Petri nets representation:	

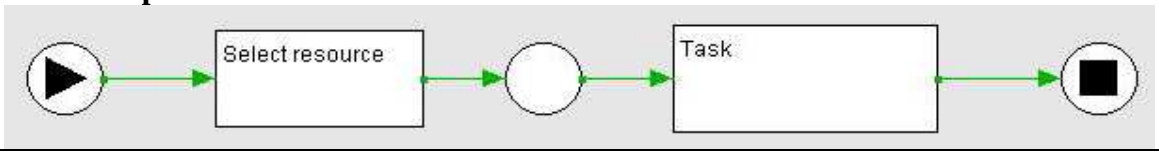


Table 4-36 Commencement on creation pattern analysis

Neither Staffware nor Websphere MQ support this pattern. The BPEL notation neither since a resource needs to accept or acquire a work item from the agenda in order to start the execution. In the other hand, BPMN and UML assume the actions/activities lived as soon as they receive the control-flow token.

Atoms does not support this pattern because its standard behaviour corresponds to the next pattern, Commencement on allocation. Then, a resource will have to wait for an allocation to execute a task.

4.2.5.2 Commencement on allocation

Identifier:	R-CA
Name of the pattern:	Commencement on allocation
Synopsis:	The ability to commence execution on a work item as soon as it is allocated to a resource.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: Work on the "Practice Tower Block Fire Drill" task commences as soon as it is allocated to a Fire Team resource.
Petri nets representation:	

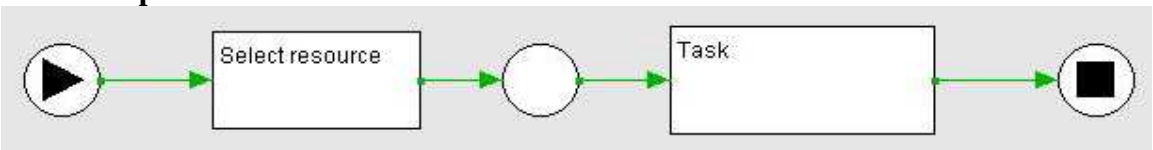


Table 4-37 Commencement on allocation pattern analysis

While Staffware do not support this pattern, Websphere MQ allows resources to configure work queues to initiate tasks on arrival. BPEL notation does not support this pattern by the same cause as last pattern whereas, BPMN and UML, do not support it because they support the last pattern.

In Atoms, a resource can execute a task since it receives the control-flow token and it is allocated to it.

4.2.5.3 Piled execution

Identifier:	R-PE
Name of the pattern:	Piled execution
Synopsis:	The ability of the workflow system to initiate the next instance of a task once the previous one has completed.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: The next "Clean Hotel Room" task can commence immediately after the previous one has finished and it can be allocated to the same Cleaner.
Petri nets representation:	

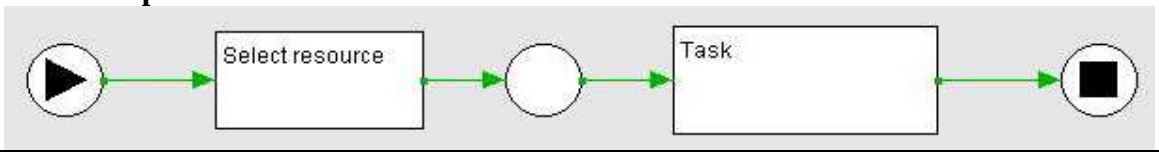


Table 4-38 Piled execution pattern analysis

Neither the commercial software studied nor the notations support this pattern.

Atoms does not support any kind of special features in execution like that.

4.2.5.4 Chained execution

Identifier:	R-CE
Name of the pattern:	Chained execution
Synopsis:	The ability of the workflow engine to automatically start the next task in case once the previous one has completed.
Context:	Apply this pattern is a good option when
Examples:	It would be a good application of the pattern: Immediately commence the next work item in the "Emergency Rescue Coordination" process when the preceding one has completed.
Petri nets representation:	

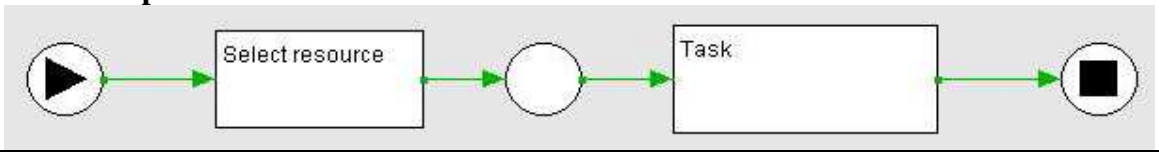


Table 4-39 Chained execution pattern analysis

Neither Staffware nor Websphere MQ support it. Whereas BPEL does not support it neither, BPMN and UML support it because when an action is completed, subsequent actions receive a control-flow token and are triggered immediately.

Atoms lets the resources to execute any available task, but the system can not start the execution of a given task.

4.3 XSL transformations from Petri nets to State charts

Once we know all the workflow resource patterns and how we have decided to transform each case, we are ready to transform the source process defined in Petri nets to the State charts specification of IMmedia Atoms. In this section, we are going to show first the specification of the output generated by our source editor, FlowiXML workflow editor, and the way that IMmedia Atoms stores its workflow processes, which will be its input too, after.

4.3.1 Output from FlowiXML workflow editor

Our workflow editor generates four different XML files as output of the workflow processes we have defined using its environment:

1. **processModel,** which represents all the transitions between the different tasks,
2. **taskModel,** which stores some information about each task and their internal execution definition,
3. **mappingModel,** which we can obtain the necessary information about the workflow resource pattern applied to each task,
4. **workflowModel,** which has stored all the information about the organization, like organizational units, user stereotypes and jobs.

Some changes have been introduced in the mappingModel file, allowing the assignment of only one pattern in the definition of a task. In the following sections is showed the content of these four XML files and the changes applied on them.

4.3.1.1 processModel

In this file we can find the information about the relations between the defined tasks of the process.

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessModel>
  <Process>
    <joinPattern type="sequential">
      <source taskId="2" taskName="Control the stock" />
      <destination taskId="4" taskName="Send order" />
    </joinPattern>
    <splitPattern type="sequential">
      <source taskId="4" taskName="Send order" />
      <destination taskId="5" taskName="Join" />
    </splitPattern>
    <joinPattern type="sequential">
      <source taskId="1" taskName="Place an order" />
      <destination taskId="3" taskName="Manage the payment" />
    </joinPattern>
    <splitPattern type="sequential">
      <source taskId="3" taskName="Manage the payment" />
      <destination taskId="5" taskName="Join" />
    </splitPattern>
    <joinPattern type="sequential">
      <source taskId="1" taskName="Place an order" />
      <destination taskId="2" taskName="Control the stock" />
    </joinPattern>
    <splitPattern type="sequential">
      <source taskId="2" taskName="Control the stock" />
      <destination taskId="4" taskName="Send order" />
    </splitPattern>
    <splitPattern type="andSplit">
      <source taskId="1" taskName="Place an order" />
      <destination taskId="2" taskName="Control the stock" />
      <destination taskId="3" taskName="Manage the payment" />
    </splitPattern>
    <joinPattern type="andJoin">
      <source taskId="3" taskName="Manage the payment" />
      <source taskId="4" taskName="Send order" />
      <destination taskId="5" taskName="Join" />
    </joinPattern>
  </Process>
</ProcessModel>
```

Inside the ProcessModel tag we find the tag Process, which includes all the transitions in the process, representing it in splitPattern and joinPattern tags. These two tags have an attribute called type, where we can define the type of split or join of the transition, which can be sequential, or “and”, “or” or “andOr” split/join. As we can see in the example, the transitions of type sequential can be repeated due to two tasks connected without any type of split or join will have two definitions of sequential transitions, one in a splitPattern tag and another one in a joinPattern tag.

4.3.1.2 taskModel

This file contains information about the execution of the task which will not used in our transformation, because of we only deal with the user interfaces related to the workflow resource patterns applied in the tasks.

```
<?xml version="1.0" encoding="UTF-8"?>
<TaskModel>
  <TaskAtProcessLevel id="4">
    <taskmodel>
      <task id="st0task0" name="Send order" type="abstraction">
        . . .
      </task>
      <enabling>
        . . .
      </enabling>
    </taskmodel>
  </TaskAtProcessLevel>
  <TaskAtProcessLevel id="3">
    <taskmodel>
      <task id="st0task0" name="Manage the payment" type="abstraction">
        . . .
      </task>
    </taskmodel>
  </TaskAtProcessLevel>
  <TaskAtProcessLevel id="2">
    <taskmodel>
      <task id="st0task0" name="Control the stock" type="abstraction">
        . . .
      </task>
    </taskmodel>
  </TaskAtProcessLevel>
  <TaskAtProcessLevel id="1">
    <taskmodel>
      <task id="st0task0" name="Place an order" type="abstraction">
        . . .
      </task>
    </taskmodel>
  </TaskAtProcessLevel>
  <TaskAtProcessLevel id="5" />
</TaskModel>
```

Inside the TaskModel tag we can define the internal behaviour of each task using the tag called TaskAtProcessLevel. We will not explain it deeply because, as we have mentioned before, we will not deal with this information of the task execution.

4.3.1.3 mappingModel

This file contains the information related to the patterns applied to defined tasks. We have modified this file on order to allow the definition of only one pattern in a task.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappingModel>
  <resourcePatterns taskId="4" taskName="Send order">
    <pattern name="R-RD" />
    . . .
  </resourcePatterns>
  <resourcePatterns taskId="3" taskName="Manage the payment">
    <pattern name="R-D" />
    . . .
  </resourcePatterns>
  <resourcePatterns taskId="2" taskName="Control the stock">
    <pattern name="R-SK" />
    . . .
  </resourcePatterns>
  <resourcePatterns taskId="1" taskName="Place an order">
    <pattern name="R-DA" />
    . . .
  </resourcePatterns>
  <resourcePatterns taskId="5" taskName="Join">
    <pattern name="R-DA" />
    . . .
  </resourcePatterns>
</mappingModel>
```

Inside the mappingModel tag we can define resourcePatterns tag for each task where we will be able to define the patterns applied in them. Inside resourcePattenrs tag we find the modification we have introduced, the tag pattern with the attribute name, filled by the identifier of the pattern applied. The original information of the applied patterns is not showed here because of we will not use it.

4.3.1.4 workflowModel

This file contains the information related to the organization. In this file there is no reflected any relation with the tasks and the process and, for this reason, we will not use it in the transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<WorkflowModel>
  <OrganizationalUnits>
    <OrganizationalUnit id="3" name="Delivery department" . . . />
    <OrganizationalUnit id="2" name="Accounting department" . . . />
    <OrganizationalUnit id="1" name="Stock control department" . . . />
  </OrganizationalUnits>
  <Jobs>
    <Job id="1" name="Warehouseman" . . . />
  </Jobs>
  <UserStereotypes>
    <UserStereotype id="1" name="Order preparator" . . . />
  </UserStereotypes>
  <RelationsJobInUnit>
    <RelationJobInUnit jobId="1" organizationalUnitId="3">
      <ContainedUserStereotype id="1" />
    </RelationJobInUnit>
  </RelationsJobInUnit>
</WorkflowModel>
```

Inside the WorkflowModel tag we find the information related to the organization mentioned before, but we are not going to explain it due to the little importance of it in this thesis.

4.3.2 XML workflow definition of IMmedia Atoms

IMmedia Atoms stores its workflow process data in an XML file called workflow.xml, which includes information about java classes that have to be included in the environment of the application, the different states of the process, and the transitions (tasks) between these states called arrows.

```
<?xml version="1.0" encoding="UTF-8"?>
<graph>
  <loader class="be.immedia.workflow.actor.computer.Null" />
  . . .
  <loader class="be.immedia.workflow.actor.human.FillTemplate" />
  <section rows="20" cols="20">
    <start id="9tmxdcka4504" row="2" col="2" />
    <state id="1patternTask1" name="Select ressource" rows="0" row="2" col="3" />
    <state id="1" name="Place an order" rows="0" row="2" col="4" />
    <end id="9tmycrk7tsfl" row="2" col="16" />
  </section>
  <arrow id="1" name="editer un document" start="1patternTask1" end="1" when="-1" class="JAVA">
    <users>
      <user id="-1">Administrateur</user>
    </users>
    <config confirm="true">
      <edit id="31" form="form: Form" />
    </config>
  </arrow>
</graph>
```

The entire process is defined inside the graph tag, the main one. Inside it, we find an assortment of java classes to include in the environment of the application, defined using the loader tags and its attribute class.

Afterwards, we can find the definition of all of the states in the process defined using the state tags inside the section tag, which has defined the number of rows and columns of the final screen. Each of these states stores information of identifier, name and the position in the screen using the different attributes of this tag. The attribute row is used when a state contains other states, but it is only a graphical support and we are not going to use it. There are two special states defined using start and end tags and they represent the start and end states of the entire process.

Then, the transitions between the states are defined using the arrow tag. The attributes of this tag are:

1. **id**, the identifier of the arrow,
2. **name**, the name of the arrow,
3. **start**, the source state of the arrow,
4. **end**, the target state of the arrow,
5. **when**, the number of minutes we can wait to execute the task (-1 implies no time limit),
6. **class**, the java class of the arrow, which defines its behaviour.

Inside the arrow tag, we can define some necessary parameters for some types of arrows. For example, if we define an arrow as `be.immedia.workflow.actor.human.Edit` (edit a document by a human resource), we have to define which users are able to execute the task in the users tag, and which form (user interface) is attached to execute it in the config tag.

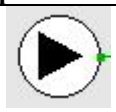
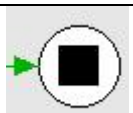
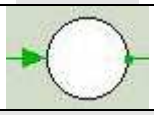
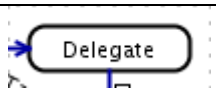
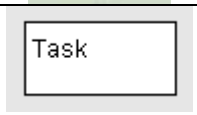
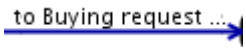
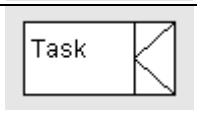
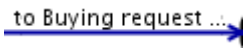
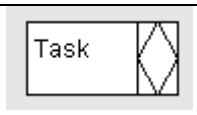
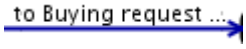
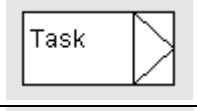
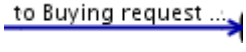
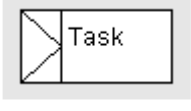
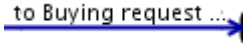
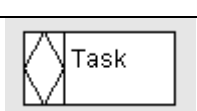
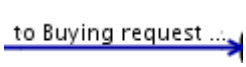
4.3.3 XSL transformation

After knowing the entire XML files format we are going to work with, we can show how the transformation has been done.

We have approached this transformation in three big steps: translate the petri nets process into a state charts one, represent the behaviour of the pattern and attach a user interface to a task when needed.

4.3.3.1 Petri nets to state charts

As we have mentioned before, petri nets defines its processes using places and tasks, whereas state charts defines them using states and transitions. To make this transformation we need to find a relation between each object in petri nets and its corresponding object (or objects set) in state charts, although not in the general specification of the UML state charts, but in the specific definition of process of the target application Atoms. The next table shows the relation found.

Petri nets	Graphical representation	State charts	Graphical representation	Conclusions
Start place		**Initial State**		These two objects represent the same in the two applications.
Final place		**Final State**		These two objects represent the same in the two applications.
Place		**State**		The state will have the name of the next task to execute.
Task (with no join no split)		**Transition**		These two objects represent the same in the two applications.
Task (with AND-split)		**Transition**		It is not possible to represent. We will advertise the workflow manager.
Task (with And/or-split)		**Transition**		It is not possible to represent. We will advertise the workflow manager.
Task (with XOR-split)		**Transition**		This behaviour is the standard one in the target application.
Task (with AND-join)		**Transition**		We can not represent any kind of AND-split. Then, we will not be necessary to represent that.
Task (with And/or-join)		**Transition**		We can not represent any kind of AND-split. Then, we will not be necessary to represent that.

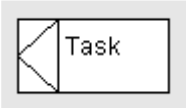
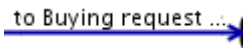
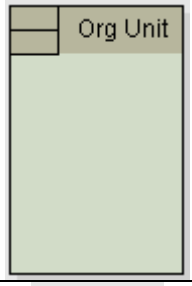
Task (with XOR-join)		**Transition**		This behaviour is the standard one in the target application.
Organizational unit		**User management**	No representation	The user management is developed to be done using the interface of the application. It could be represented as a user group.
Resource box		**User management**	No representation	The user management is developed to be done using the interface of the application. It could be represented as a user or a user group.

Table 4- 40 Summary of the relations between object from FlowiXML workflow editor and Atoms

Table 4-40 shows us some conclusions about the relation found, but we will complement it with a bigger definition of each relation defined as follows:

1. Both **start place** and **end place** have a direct transformation into start state an end state. These two objects do not present any problem during the transformation.
2. **Places** will be represented as states. Places do not have name whereas states have it, so the name of the states will be defined in the patterns definition (see section 4.3.3.2).
3. **Tasks** will be defined as transitions, which will be expanded depending on the pattern applied to the task. As we have seen in former sections (see section 4.1), a task which has applied a direct allocation pattern (see Figure 4-9) will be transformed in a set of states and transitions that represents the behaviour of the applied pattern (see Figure 4-10). The real task will be represented as the transition called "TASK", whereas the other transitions will represent the desired behaviour.

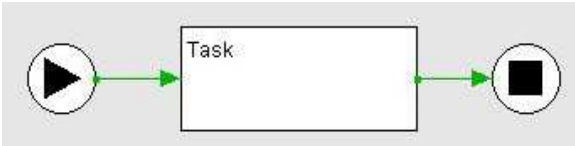


Figure 4-9 Task with a direct allocation pattern applied

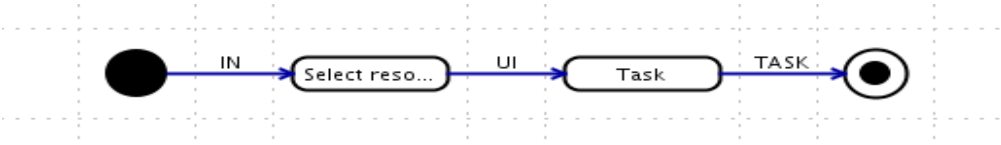


Figure 4-10 Representation of a task with direct allocation pattern applied

4. **Tasks** defined with any kind of **AND-split** will not have the desired behaviour once the transformation is done. The target application does not allow two different paths to follow in a process at the same time, so the best solution found is to advertise the workflow manager about this kind of split and he will be able to take the best decision for each transformation. It is not possible to translate petri nets into state charts due to this kind of splits [Eshu05], but in our case we have a problem added, the restriction of only one document created for process. One possible solution thought was to connect the end of the different paths to the beginning of the other paths, allowing the users to

follow one way and return to the beginning of another when finished (see Figure 4-11), but this solution could not make understandable the process transformed if the process has a lot of this kind of splits (see Figure 4-12, green arrows).

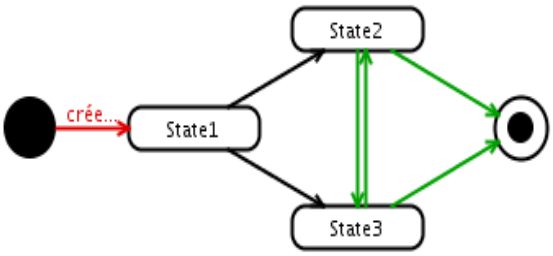


Figure 4-11 Possible solution to AND-split problem

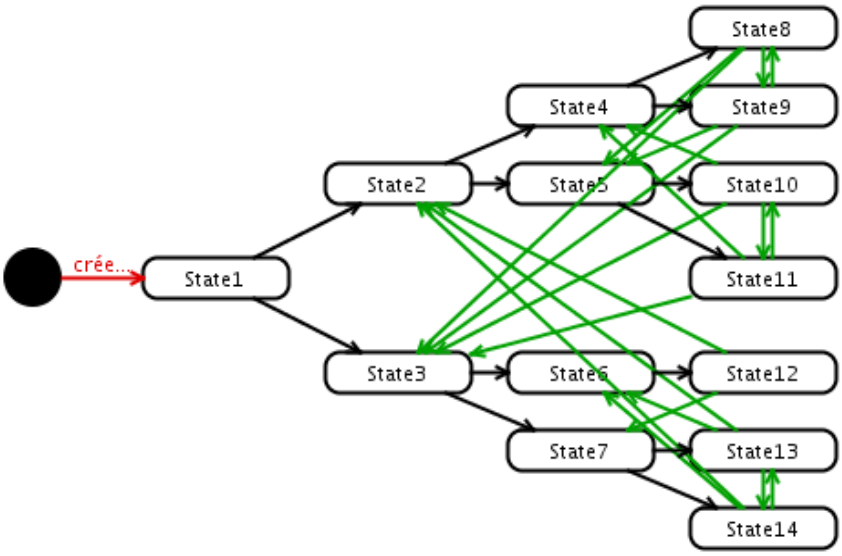


Figure 4-12 Conflictive process transformation when an AND-split is found

5. **Tasks** defined with **OR-splits** will be transformed without any special attention because of they represent the standard behaviour of the target application in the transaction splitting.
6. **Tasks** defined with any kind of **join** will be represented as OR-join tasks. The problem about the representation of the AND-split does not allow us to represent the AND-join neither.
7. **Organizational units and resource boxes** will not be represented in the target application. The concept of organizational unit does not exist there, whereas the resource boxes could be transformed in users. In this thesis, we will not deal with the user management in the target application, so only one user, the administrator, will be able to execute the entire process.

4.3.3.2 Patterns representation

Each task defined in the source editor tool will have a workflow resource pattern applied. Some of these patterns will have the same representation due to the incompatibility of the source and target applications.

Four different pattern representations have been defined to represent the entire set of workflow resource patterns.

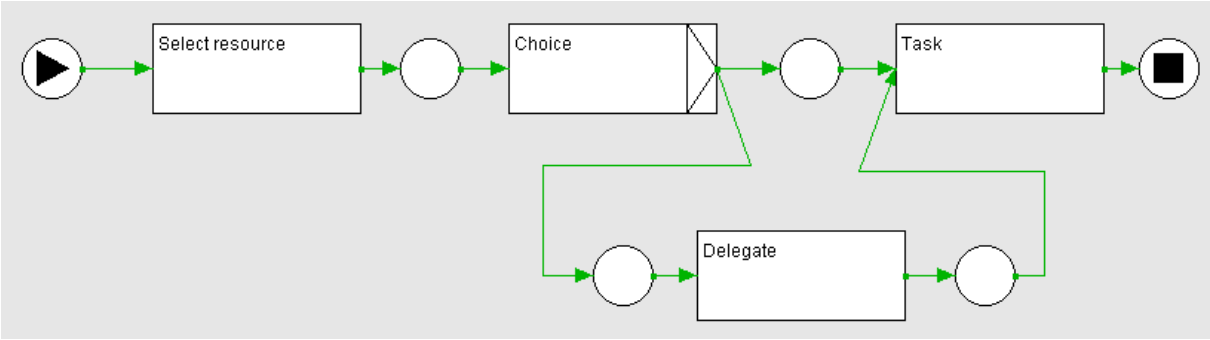


Figure 4-13 Reallocation patterns definition

Figure 4-13 represents the definition of the different reallocation patterns. In this set of patterns we found the following ones: delegation, escalation, stateful reallocation and stateless reallocation. We can not represent the difference between the stateful and stateless reallocations in Atoms, so these two patterns will have the same representation.

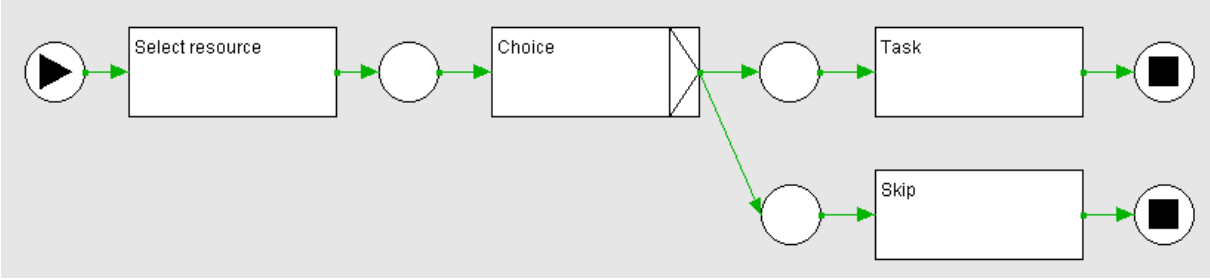


Figure 4-14 Skip pattern definition

Skip pattern is a particular case of pattern, not similar to another one, and it will have a special representation (see Figure 4-14).

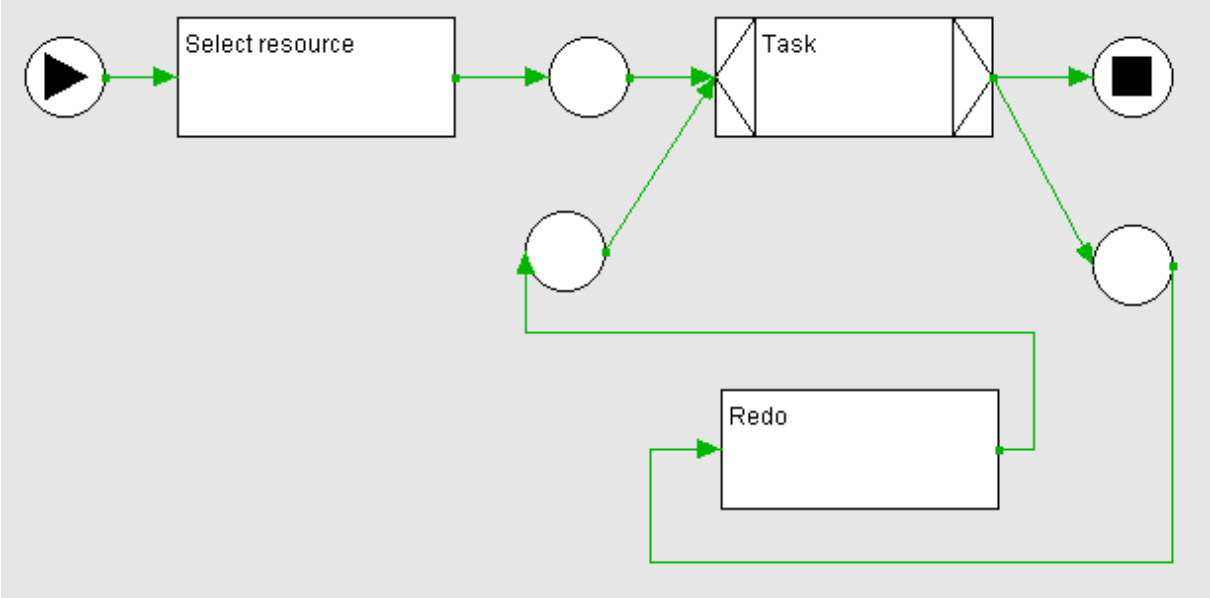


Figure 4-15 Redo pattern definition

As skip pattern, redo pattern will have a particular representation in the transformation (see Figure 4-15).

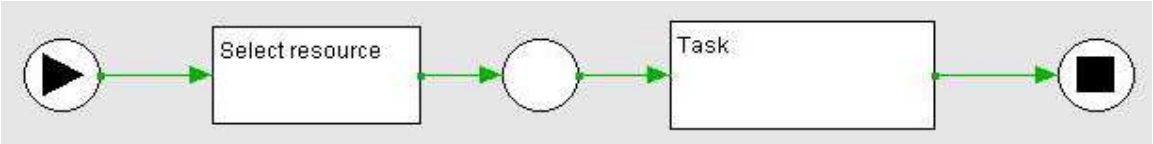


Figure 4-16 Direct allocation pattern definition

The rest of patterns we have not mentioned will have the representation of the direct allocation pattern (see Figure 4-16), maybe due to it represents their behaviour or maybe due to the impossibility to represent them using the Atoms features as we have explained before (see section 4.2). In case of our transformation can not represent one of this pattern and it represents it like that, the transformation will advertise the workflow manager of the lack of correctness in the result. It will be necessary human management to accomplish the behaviour represented in the source workflow editor.

4.3.3.3 User interfaces

At this point, we only need to attach the user interfaces needed to each transition created. We will use two kinds of user interfaces, one to select the resource to execute a task and another to go to the next state unconditionally (see Figure 4-17).

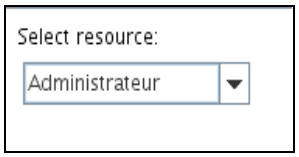


Figure 4-17 User interface to select a resource

As a result of the transformation of the process we will obtain a bigger process (bigger number of states) due to the transitions added to represent the patterns behaviour. For each task we will obtain a set of states, whose transitions will have to be defined in these two ways: select a resource or go to the target state unconditionally. We are going to show an example of application of the four possible representations, using the transformation of the second case study of the next chapter (see section 5.2).

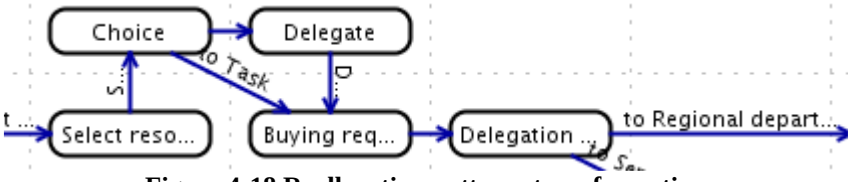


Figure 4-18 Reallocation patterns transformation

Figure 4-18 shows the transformation of a task called Buying request approval defined with a delegation pattern applied. During its execution, we will arrive at the Select resource state and we will be able to execute the selection resource task using the user interface last mentioned. After that, we will arrive at the Choice state, where we will have to decide if we want to execute the task or to reallocate it to another resource using unconditional transitions. If we decide to execute the task, we will be in the state called Buying request approval in this case, and we will have to execute it, using the specific task form, to arrive to the Delegation pattern end, which is used to redirect the control flow of the process to the next task. In the other

hand, if we would have decided to reallocate the task, we would have to select another resource using the same user interface as before to arrive to the main state too.

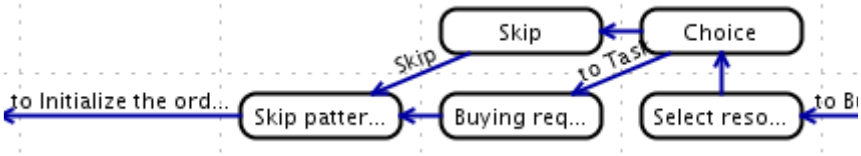


Figure 4-19 Skip pattern transformation

Figure 4-19 shows us the transformation of a task called Buying request responsible approval defined with a skip pattern applied. As the last example, we arrive at Select resource state and we have to select the resource using the user interface mentioned before to arrive at Choice state. Once there, we will be able to choose if we want to skip the task or to execute it. If we want to execute it, the process will be the same as the reallocation patterns explained before till the Skip pattern end state but, if we want to skip the task, we will arrive at the Skip state and after we will execute an unconditional transition to finish at the Skip pattern end too.

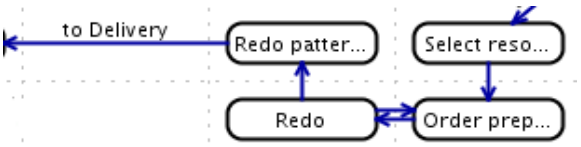


Figure 4-20 Redo pattern transformation

Figure 4-20 shows us the transformation of a task called Order preparation defined with a redo pattern applied. We will always start in a Select resource state and we will execute its transition using the user interfaces before explained. After that we will arrive at the Order preparation state, when we will only be able to execute the task using its specific user interface. Then, we will arrive at Redo state, when we will have to decide if we want to redo the task or to finish it going unconditionally to the Redo pattern end state. If we decide to redo the task, we will return to the Order preparation state and the behaviour in this state will be the same as before.

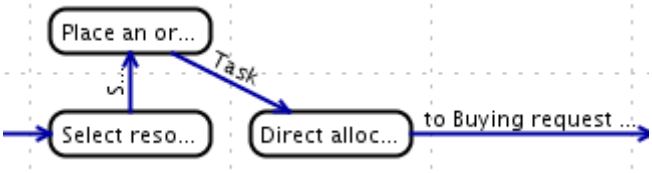


Figure 4-21 Direct allocation patterns transformation

Figure 4-21 shows us the transformation of a task called Place an order defined with a direct allocation pattern applied. As the other transformations, the execution will start in the Select resource state and its transition will be executed using the same user interface used before to select a resource. Afterwards we will arrive at the Place an order state whose transition will be executed using the specific user interface of the task to finally arrive to the Direct allocation end state, where the transformation finishes.

Once we know the essential steps of this transformation we can show complete case studies to understand it better in the next chapter.

Chapter 5 Case studies

This chapter will show two case studies allowing the reader to understand better how the transformation is done.

The first one will represent a process that an order placed in a current web shop could trigger. This little case study will be detailed as much as possible showing the source process, the transformed process and the user interface used to execute it.

Afterwards, another one will represent a bigger process about a buying request in a company which produces and manages the order it receives. This process will not be detailed as the first one, but it will be possible to know what patterns have been applied in each task and the process defined in petri nets, and after its transformation, will be shown.

5.1 Case study 1 - Simplified buying request

This case study is about a simplified buying request in an online shop. This business process (see Figure 5-1) starts when a customer places an order. At this point, we have defined an AND-split, starting two different (and concurrent) processes: the one related to the accounting department and another one related to the delivery departments and its previous control of the stock in the stock control department. Afterwards, they join in a task defined only to unify the two processes, we have called join.

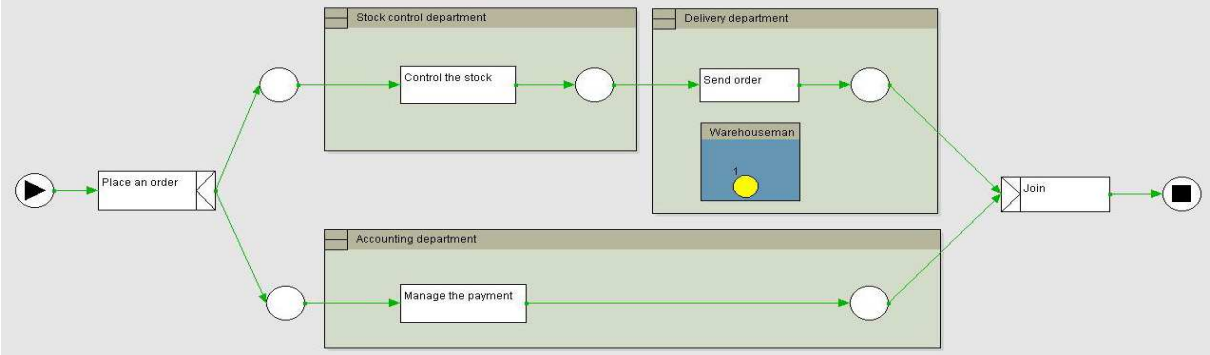


Figure 5-1 Process of a simplified buying request in an online shop

Referring to the workflow resource patterns, we have applied different patterns to each task according to their place in the different departments of the organization and the execution of the tasks. The patterns applied for each task are defined as follows:

1. **Place an order.** This task is always executed by the customer, and for this reason, we have applied the direct allocation pattern.
2. **Control the stock.** This task is executed by the stock controllers. We have applied the skip pattern, to allow them the option of skip this task, because of maybe they have controlled the stock of a given good before for another task, knowing surely that there is no any lack of stock to satisfy the actual order.
3. **Send order.** This task is executed by the warehousemen. We have applied the shortest queue pattern to assign the task to the warehouseman with less charge of work.
4. **Manage the payment.** This task is executed by the accountants. We want to allow the accountants to reallocate this task to other workers if the have a big charge of work, and for this reason we have applied a delegation pattern.

Once the process is defined and we want to transform it, we have to know what kind of transformation has each of the applied patterns:

1. The place an order task will have the simplest pattern representation, adding a task to select a resource before the execution of the main task.
2. The control the stock task representation has to allow us the option of skip the task. So, before selecting the resource is going to execute the task, we have to add a task that allows us to choice if we want to execute the task or we want to skip it. Afterwards, they join in the same task.
3. The send order task will be allocated to the warehouseman with less charge of work. This feature can not be represented in Atoms, so we will apply the direct allocation pattern and the workflow manager will have to be carefully choosing the warehouseman to execute this task.

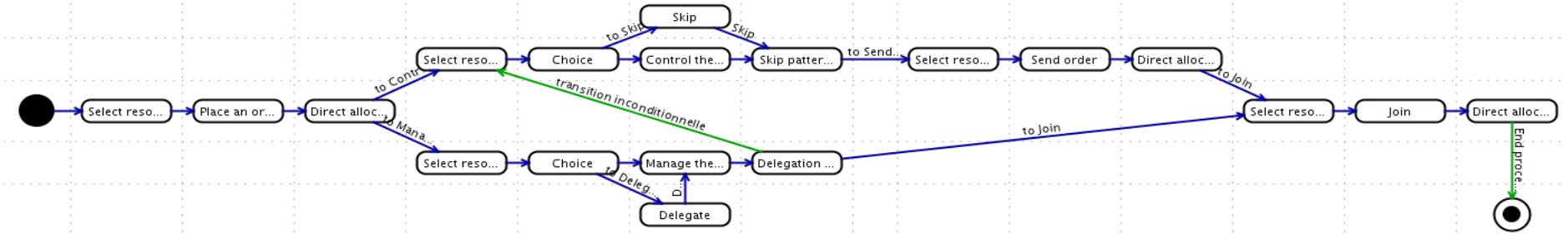


Figure 5-2 Simplified buying request process transformed to state charts in Atoms

4. The manage the payment task will have to allow us the reallocation of the task to another accountant. In this case, we will add a task that allows the accountant to choose if he wants to execute the task or he prefers to delegate it to another accountant.

Once we have the transformation done, we notice that in the petri nets process we had applied an AND-split in the place an order task. As we know, we can not represent this kind of splitting in the state charts specification of Atoms. In this case, we have to choose what way will be executed first and, knowing the case the process covers, we have to manage the payment before starting to prepare the order, controlling the stock and sending it. For this reason, we have added a transition between the last task of the manage the payment pattern tasks to the first of the control the stock pattern tasks (see Figure 5-2).

In terms of user interfaces, once the behaviour is well-represented in state charts, for the selection of the resources we will use a simple interface where we will choose the resource to execute the actual task in a combo box (see Figure 5-3).

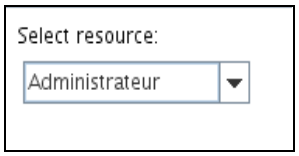


Figure 5-3 Use interface of the select resource pattern task

5.2 Case study 2 - Complete buying request

This case study is about a complete buying request in a company. As case study 1, this process starts when a customer (or an external company) places an order. Afterwards, the commercial department first, and the warehouses classify in regional departments after, execute a series of tasks to prepare and deliver it (see Figure 5-4).

We are not going to focus on the process definition in this case study but we will show the patterns applied and why they have been applied:

1. **Place an order.** This task has a direct allocation pattern applied due to only the customer or an external company can execute it.
2. **Buying request approval.** This task has a delegation pattern applied to allow the reallocation of this task to another resource if the user has allocated it can not execute it.
3. **Send buying request cancellation.** This task has a retain familiar pattern applied because of we want that the same worker who has executed the previous task executes it.
4. **Regional department assignment.** This task has a history-based allocation pattern applied because of the company wants to have the same regional department assigned in the different orders of the same customer.
5. **Responsible assignment.** This task has a direct allocation pattern to manage strictly the resources who can execute this task.
6. **Buying request warehouse approval.** This task has a skip pattern applied because of, as the order has been approved by the commercial department and by the regional department before, the responsible of the warehouse can skip it if he has a good experience with the customer and he does not think that another approval is necessary.
7. **Initialize the order preparation.** This task has a shortest queue pattern applied. The reasons are that using this pattern, we can have the tasks to do well-distributed between the workers.
8. **Production request.** This task has a retain familiar pattern applied due to the company wants that the worker has initialized the order before execute this task after.
9. **Production.** This task has a shortest queue pattern applied. The reasons are the same as in the initialize the order task.
10. **Order preparation.** This task has a redo pattern applied because of an order can be bad-prepared and we want to allow the workers to redo it.
11. **Delivery.** This pattern has a direct allocation pattern because of this task will be executed by an external post company and it is not in charge of our company.
12. **Satisfaction report.** This pattern has a direct allocation pattern applied due to we want that the customer will execute this task.

Figures 5-4 shows us the source process defined in petri nets and its transformation can be seen in Figure 5-5.

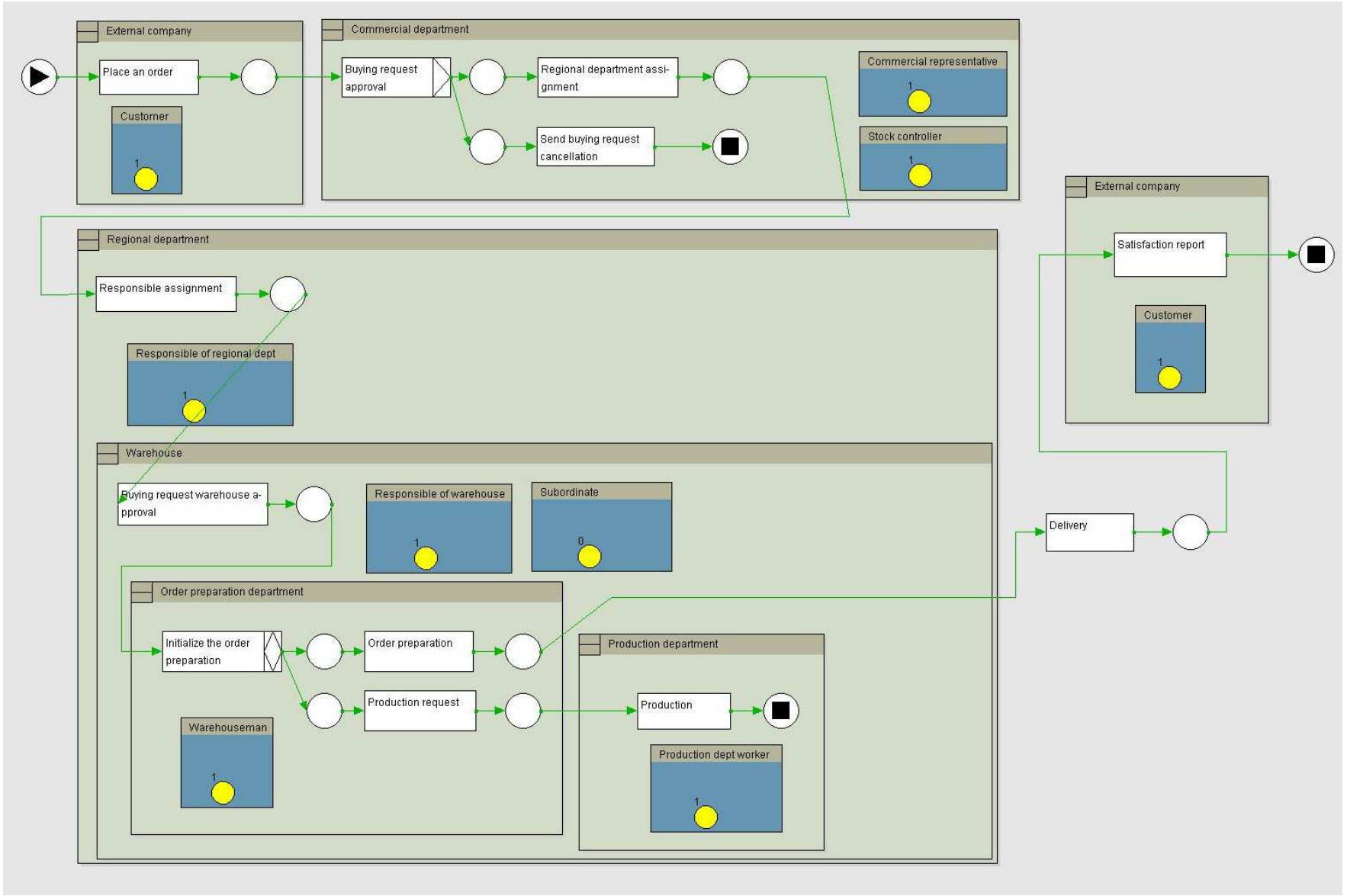


Figure 5-4 Process of a complete buying request in a company

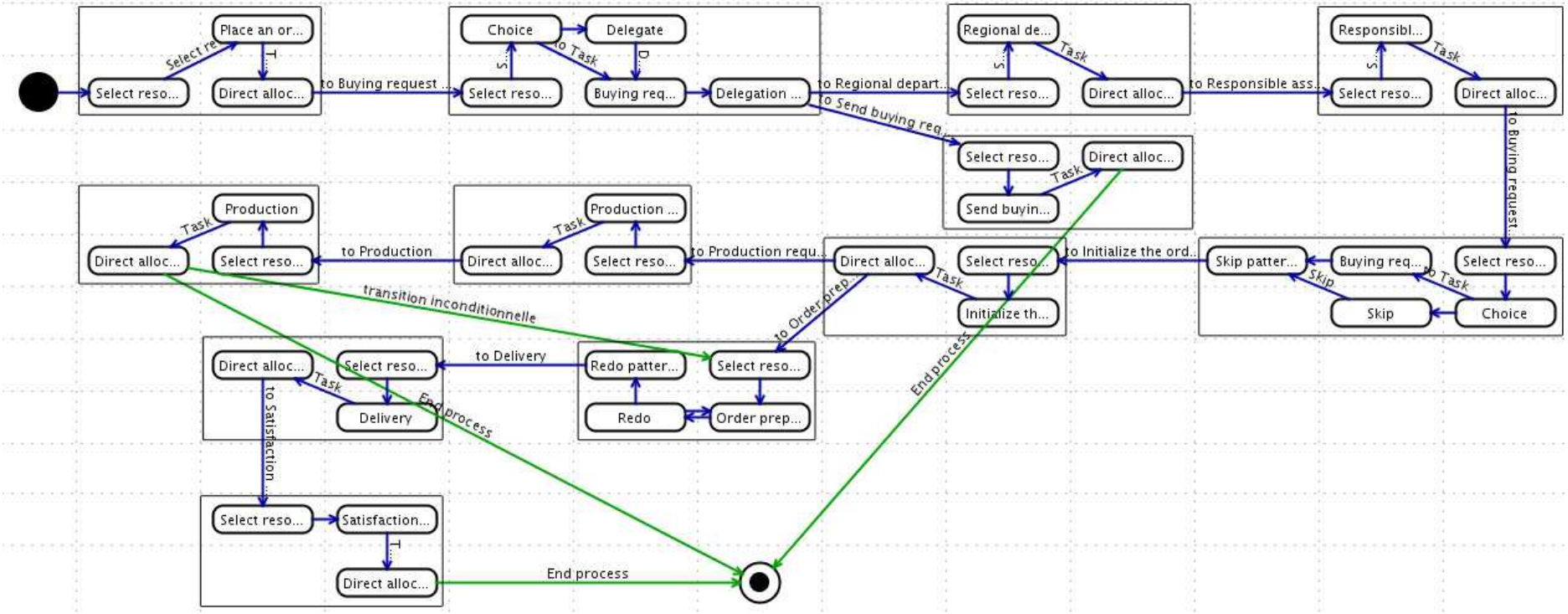


Figure 5-5 Complete buying request process transformed in state charts in Atoms

Chapter 6 Conclusion

This thesis concludes with this chapter.

It starts with a summary of the main contributions of this thesis. Afterwards, an explanation of some future work which could bring this problem to new frontiers of solution will be presented.

6.1 Contribution

This thesis has dealt with the current need, within this generation, for user interfaces in a workflow information system, which allows the definition of workflow resource patterns in its tasks. For this purpose, commercial software has been chosen to target workflow management systems, called Atoms. Each workflow resource pattern has been studied in order to find a valid transformation. Whilst knowing the features of the target application, it is impossible to represent, in its entirety, these set of patterns. The most common commercial workflow management systems do not support, for the most part, the behaviour workflow resource patterns because of they have only appeared recently. Furthermore, some of this software has been well-consolidated in the business world for a long time. Some others initiatives, such as YAWL, have been exclusively developed to support them, but in this thesis, it was intended to link FlowiXML workflow editor with a commercial software, which is unable to support a large amount of the workflow resource patterns, and find the best solution to each of them.

6.2 Future work

As mentioned throughout this thesis, the resource management in the target application has not been implemented. This is the main work still to be done in order to progressively improve this transformation; providing it with different groups of users, making possible the allocation to different resources using the user interfaces generated automatically. Atoms is not developed to manage users with regards to their external environment, so to reach this goal, without knowing the data base management system which Atoms uses, it is necessary to study the different relations between the data base tables, they create. Moreover, Atoms could be modified in order to simplify the data base management from its exterior. Currently, this is seen as complicated therefore, this modification could interest IMmedia S.A. for future releases.

Another project could study the patterns which have not been possible, with respect to their behaviour, dealing directly with the IMmedia S.A. staff to find which pattern could be useful to implement on Atoms. As we have seen throughout this thesis, the most common commercial workflow management systems do not support the great majority of the workflow resource patterns. Therefore this project could study which patterns are; most popular, useful in the business world. If the developers of Atoms agree to implement some new features to its application, the transformation could be upgraded to a new version that includes the new supported patterns transformation.

This transformation could be done another time using another target application. In this thesis, some problems have been found working with petri nets and state charts transformation. Therefore, another transformation could be done, using a petri nets based workflow definition system, trying to find the support of the majority of the workflow resource patterns. In this way, the transformation will be more compatible to the source and transformed process.

Finally, a project that deals with the transformation of the FlowiXML workflow editor output to be used in a YAWL-supported system could be done. This would allow for perfect behavioural representation of the different patterns, since YAWL has been developed directly to support all workflow patterns.

References

A

[Abra99]
Abrams, M. Phanouriou, C., Batongbacal, A.L., Williams, S., & Shuster, J. UIML: An Appliance-Independent XML User Interface Language. In A. Mendelzon, editor, Proceedings of 8th International World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999), Amsterdam, 1999. Elsevier Science Publishers.

[Agil08]
http://www.agilemodeling.com

[Azev00]
Azevedo, P., Merrick, P., and Roberts, D. OVID to AUIML user oriented interface modeling. In Proceedings of 1st International Workshop Towards a UML Profile for Interactive Systems Development TUPIS00. York, October 2000.

B

[Bosw02]
Boswell, D., King, B., Oeschger, I., Collins, P., and Murphy, E. Introduction to XUL. In "Creating Applications with Mozilla", O'Reilly, Sebastopol, September 2002.

[Bpmn08]
http://www.bpmn.org

[BuiG08]
http://www.buigallery.com/

C

[Carl62]
C. Petri. Kommunikation mit Automaten, Dissertation, Rheinisch-Westfalisches Institut fur. Intrumentelle Mathematik an der Universitat Bonn, Bonn. 1962.

[Cui05]
Cui, X., Transforming Phone-based Interfaces for Web Access: from WML to UsiXML, Louvain-la-Neuve, 2005.

D

[Desa08]
http://www.desarrolloweb.com

[Dena08]
http://www.denali.be

E

[Eich04]
Eichholz, C., Dittmar, A., Forbrig, P. (2004) Using Task Modeling Concepts for Achieving Adaptative Workflows. Proceedings of DSV-IS-EHCI'94, Springer-Verlag, Berlin, LNCS,3425.

[Eshu05]
Eshuis, R. Statecharting Petri Nets. BETA Working Paper Series WP 153, Eindhoven University of Technology, 2005.

[Espo05]
Esposito, D. Getting Started with Microsoft Windows Workflow Foundation: A Developer Walkthrough. September 2005. http://msdn.microsoft.com/winfx/reference/workflow

G

[Gome04]

Gomes de Sousa, L., and Leite, J.C. XICL- An Extensible Mark-up Language for Developing User Interface and Components. Proceedings of the Fifth International Conference on Computer-Aided Design of User Interface CADUI'2004.

[Guer06]

Guerrero, J., Conceptual Modeling of User Interfaces to Workflow Information Systems, Louvain-la-Neuve, 2006.

L

[Lema07]

Lemaigre, C., Développement d'un éditeur graphique de workflow générant automatiquement ses spécifications fonctionnelles, Louvain-la-Neuve, 2007.

M

[Mano02]

Dragos A. Manolescu., An Extensible Workflow Architecture with Objects and Patterns. TOOLSEE 2001, March 2002, Sofia, Bulgaria.

[Mars94]

Marshak, R.T., Workflow White Paper – An overview of Workflow Software, Workflow'94, San Jose, 1994.

[Mars97]

Marshak, R.T. Workflow: Applying Automation to Group Process. In Coleman, D. (ed.): Groupware-Collaborative Strategies for Corporate LANs and Intranets. Prentice Hall PTR, 1997, p.p. 143-181.

[Mura89]

Murata, T., Petri nets: properties, analysis and applications, Proceedings of the IEEE, 77(4), 541-80, April 1989.

O

[Orac09]

http://www.oracle.com

P

[Puer02]

Puerta, A., and Eisenstein, J. XIML: A common representation for interaction data. In Proceedings of the 7th International Conference on Intelligent User Interfaces, pp. 69-76. ACM Press, January 2002.

S

[Souc03]

Souchon, N. and Vanderdonckt, J. A review of XML-compliant user interface description languages. DSV-IS2003, 2003.

[Spar08]

http://www.sparxsystems.com

[Stav04]

Stavness, N. and Schneider, K. (2004) Supporting Flexible Business Processes with a Progression Model. Workshop: Making Model-based UI Design Practical: Usable and Open Methods and Tool.

T

[Tibc09]

http://www.tibco.com

U

[UsiX08]

http://www.usixml.org/

V

[vand98]
van der Aalst, W.M.P. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21--66, 1998.

[vand01]
van der Aalst, W.M.P. and Kumar, A. XML Based Schema Definition for Support of Inter-organizational Workflow. University of Colorado and University of Eindhoven. Technical Report, 2001

[vand03]
van der Aalst W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P., Workflow Patterns. Distributed and Parallel Databases, 14(3), July 2003, pages 5-51.

[vand05]
van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30 (2005) 245—275

W

[W3C09]
http://www.w3.org

[W3Sc09]
http://www.w3schools.com/

[WfMC99]
WfMC (1999) Terminology & Glossary. Workflow Management Coalition. Document Number WFMC-TC-1011. Document Status – Issue 3.0. Feb-99.

[Wiki09]
http://www.wikipedia.com

[Work08]
http://www.workflowpatterns.com

Y

[YAWL08]
http://yawlfoundation.org/

Annex. Attached content

In the attached disc we can find the source code of the XSL transformation called UIGenerator.xsl, which can be executed using the java project included, called SampleXalan, amongst other execution options. This project is included because is the used one throughout the tests done for this thesis and to facilitate the execution of it. To use that, we need to define the SampleXalan folder as workspace of an Eclipse project and add a folder called xml, which will include the output XML files of the FlowiXML workflow editor and the UIGenerator.xsl source code, but also the four pattern definitions. In the folder added, all of these steps have been done and is not necessary to do it again.

The UIGenerator.xsl source code file and this thesis in PDF format are inside the attached content too.