

*new*YAWL: Specifying a Workflow Reference Language using Coloured Petri Nets*

Nick Russell¹, Arthur H.M. ter Hofstede² and Wil M.P. van der Aalst^{1,2}

¹Eindhoven University of Technology,
PO Box 513, 5600MB, Eindhoven, The Netherlands
`{n.c.russell,w.m.p.v.d.aalst}@tue.nl`

²Queensland University of Technology,
PO Box 2434, QLD, 4001, Australia
`a.terhofstede@qut.edu.au`

Abstract. *new*YAWL is a business process modelling language founded on the workflow patterns. It radically extends the YAWL offering to provide holistic support for the control-flow, data and resource perspectives and allows business processes to be captured in sufficient detail that they can be directly enacted. In order to ensure that business processes are executed in a deterministic way, *new*YAWL is based on formal foundations. This paper describes the approach taken to specifying the operational semantics for *new*YAWL based on *Coloured Petri Nets*. It discusses the development of the semantic model for *new*YAWL, which was undertaken using *CPN Tools*, and the experiences associated with developing a complete operational design for an offering of this scale using formal techniques.

1 Introduction

Over recent years the concept of the business process has garnered increasing interest as organisations seek to better understand what they do and how they can do it more efficiently. Business processes are increasingly viewed as corporate assets which companies must manage and maintain if they are to continue to operate effectively. In response to the rising demand for flexible means of automating parts of (or even entire) business processes, the field of workflow technology underwent explosive growth during the 1980s and 1990s as organisations sought configurable forms of process support.

As with many early stage technologies, individual workflow offerings provided a distinct approach to modelling the processes that they sought to automate, thus obviating any potential for standardising the representation and enactment of processes or integrating processes based on distinct offerings. Attempts by industry bodies such as the Workflow Management Coalition (WfMC) (e.g. [Wor95]) to resolve this impasse have only been marginally successful in establishing broadly adopted modelling formalisms. Consequently the majority of guidance in this area has come from

* This research is conducted in the context of the *Patterns for Process-Aware Information Systems (P4PAIS)* project which is supported by the Netherlands Organisation for Scientific Research (NWO). It is also receives partial support from the Australian Research Council under the Discovery Grant *Expressiveness Comparison and Interchange Facilitation between Business Process Execution Languages*.

one of two sources: (1) individual workflow offerings or models, such as MOBILE, WIDE and XPDL 2.0, that support a comprehensive range of concepts that generalise well to other workflow initiatives and (2) the enterprise modelling field, which includes techniques such as the Zachmann framework, EKD, IDEF, CIMOSA and ARIS, that seek to characterise the range of concepts that are relevant to modelling an organisation and its constituent processes and provide integrated approaches to capturing this information. Whilst there is general agreement across most of these offerings that a comprehensive business process model should include consideration of (at least) the control-flow, data and resource (or organisational) aspects of a business process, there is a wide variation in the range of concepts that individual techniques support for each perspective. Moreover, none of the popular modelling notations are based on a formalised model, thus leaving open the potential for ambiguity when capturing and enacting a business process.

An alternate approach to identifying the range of constructs that should be supported in a business process modelling language can be pursued which is based on *patterns*. By definition, patterns identify meaningful constructs that exist in a given problem domain. The *Workflow Patterns Initiative*¹ has established a catalogue of patterns that are relevant to the domain of business process modelling and enactment through a comprehensive evaluation of workflow and case handling systems, business process modelling and execution languages and web service composition standards. Proposed by van der Aalst et. al [AHKB03] in an effort to characterise the desirable properties of workflow languages, this research initially focussed on the control-flow perspective and identified 20 patterns which described “generic, recurring constructs” [RZ96]. The ubiquity of the patterns was soon recognised and catalogues of patterns have also been developed for the data [RHEA05] and resource [RAHE05] perspectives.

In this paper we propose *newYAWL*, a comprehensive workflow reference language based on formal foundations. *newYAWL* is founded on the workflow patterns ensuring that it recognises current practice in the process technology field and supports the capture and enactment of a wide range of workflow constructs in a deterministic way. The formalisation of *newYAWL* is based on Coloured Petri Nets [Jen97] thus providing a precise definition of the operational semantics of the *newYAWL* language that can be directly executed. The complete formalisation of a comprehensive workflow language encompassing multiple perspectives is a complex activity as demonstrated by the resultant size of the semantic model for *newYAWL* which incorporates 55 distinct pages of CPN diagrams and encompasses 480 places, 138 transitions and in excess of 1500 lines of ML code. Nevertheless, the development of the semantic model effectively demonstrates that with the correct specification tools, it is possible to formally define languages of this scale.

The paper proceeds as follows: Section 2 introduces the various constructs that make up the *newYAWL* workflow language. Section 3 describes the content of the *newYAWL* business process language and the manner in which it is captured and operationalised. Section 4 overviews related work. Section 5 discusses the experiences associated with the formalisation of *newYAWL* and concludes the paper.

¹ Further details on the workflow patterns, including detailed definitions, product evaluations, animations, vendor feedback and an assessment of their overall impact can be found at www.workflowpatterns.com.

2 *new*YAWL: A patterns-based workflow language

The workflow patterns triggered the development of *YAWL* [AH05] — an acronym for *Yet Another Workflow Language*. Unlike other efforts in the BPM area, *YAWL* sought to provide a comprehensive modelling language for business processes based on formal foundations. The content of the *YAWL* language is an adaptation of Petri Nets informed by the workflow patterns. One of its major aims was to show that a relatively small set of constructs could be used to directly support most of the workflow patterns identified. It also sought to illustrate that they could coexist within a common framework. In order to validate that the language was capable of direct enactment, the *YAWL System*² was developed, which serves as a reference implementation of the language. Over time, the *YAWL* language and the *YAWL System* have increasingly become synonymous and have garnered widespread interest from both practitioners and the academic community alike³.

Initial versions of the *YAWL System* focussed on the control-flow perspective and provided a complete implementation of 19 of the original 20 patterns. Subsequent releases incorporated limited support for selected data and resource patterns, however this effort was hampered by the lack of a complete formal description of the patterns in these perspectives. Moreover, a recent review [RHAM06] of the control-flow perspective identified 23 additional patterns which illustrate a number of commonly used control-flow constructs, many of which *YAWL* is unable to provide direct support for, including the partial join, transient and persistent triggers, iteration and recursion.

In an effort to manage the conceptual shortcomings of *YAWL* with respect to the range of workflow patterns that have now been identified, a substantial revision of the language is proposed — termed *newYAWL* — which aims to support the broadest range of the workflow patterns in the control-flow, data and resource perspectives. *newYAWL* provides a comprehensive formal description of the workflow patterns, which to date have only partially been formalised. It has a complete abstract syntax which identifies the characteristics of each of the language elements. Associated with this is an executable, semantic model for *newYAWL* — presented in the form of a Coloured Petri Net — which defines the runtime semantics of each of the language constructs. The following sections provide an overview of the features of *newYAWL* in the control-flow, data and resource perspectives.

2.1 Control-flow perspective

Figure 1 identifies the complete set of language elements which comprise the control-flow perspective of *newYAWL*. All of the language elements in *YAWL* have been retained and perform the same functions. A more detailed discussion of *YAWL* can be found in [AH05]. Several new constructs have been added based on the full range of workflow patterns that have now been identified. These are:

² See <http://www.yawl-system.com> for further details of the *YAWL System* and to download the latest version of the software.

³ Hereafter in this paper, we refer to the collective group of *YAWL* offerings developed to date — both the *YAWL* language as defined in [AH05] and also more recent *YAWL System* implementations of the language based on the original definition (up to and including release Beta 8.1) — as *YAWL*.

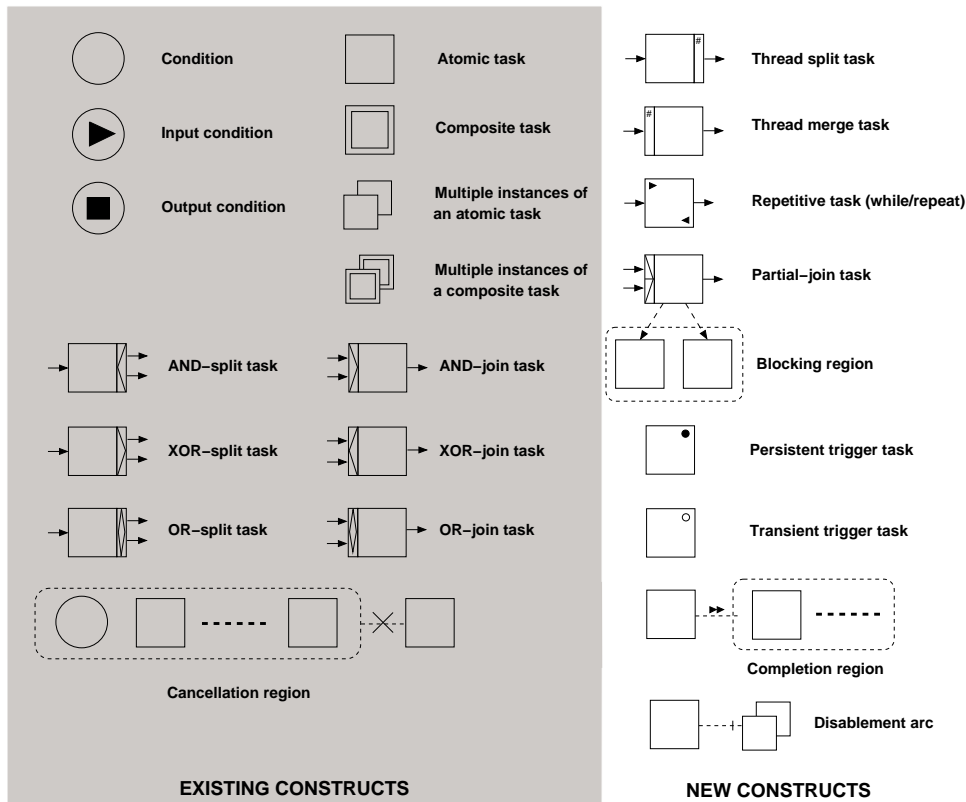


Fig. 1. *newYAWL* symbology

- the *Thread split* and *Thread merge* constructs, which allow the thread of control to be split into multiple concurrent threads or several distinct threads to be merged into a single thread of control respectively. The number of threads being created/merged is specified for the construct in the design-time model. Figure 2(a) illustrates these constructs. After the *make box* task, twelve threads of control are created ensuring that the *fill bottle* task runs 12 times before the *pack box* task can run (merging these threads before it commences);
- the *Partial join* (also known as the m-out-of-n join) allows a series of incoming branches to be merged such that the thread of control is passed to the subsequent branch when m of the incoming n branches are enabled. In Figure 2(b), the *cancel booking* task has a 1-out-of-3 join associated with it. If any of the incoming branches are enabled, then the *cancel booking* task is enabled (and any preceding tasks that are still executing in the associated cancellation region are withdrawn);
- the *Structured loop* (which supports while, repeat and combination loops) allows a task (or a sequence of tasks in the form of a subprocess) to execute repeatedly based on conditional tests at the beginning and/or end of each iteration. The loop is structured in form and it has a single entry and exit point. Figure 2(c) illustrates a repeat loop for the *check backup* task which executes repeatedly until all backups have been verified (i.e. it is a post-tested repeat loop);

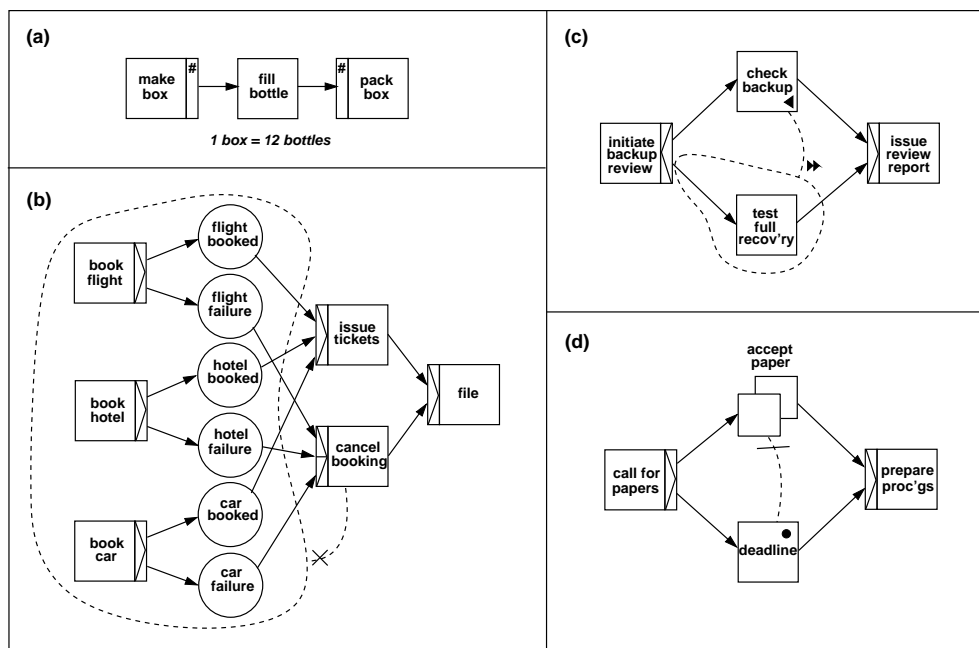


Fig. 2. Examples of *newYAWL* control-flow constructs

- the *Completion region* supports the forced completion of tasks which it encompasses. In Figure 2(c) the *test full recovery* task is forcibly completed once (all iterations of) the *check backup* task has finished. This allows the *issue review report* task to be immediately enabled;
- *Persistent triggers* and *Transient triggers* support the enablement of a task being contingent on a trigger being received from the operating environment. They are durable or transient in form respectively. Figure 2(d) illustrates a persistent trigger (assumedly associated with some form of alarm) which allows the *deadline* task to be enabled when it is received. As this trigger is durable in form, it is retained for future use if it is received before the thread of control arrives at the *deadline* task;
- the *Disablement arc* allows a dynamic multiple instance task to be prevented from creating further instances but allows for each of the currently executing instances to complete normally. Figure 2(d) has a disablement arc associated with the *deadline* task which prevents any further papers from being accepted once it has completed.

2.2 Data perspective

Whilst the control-flow perspective has received considerable focus in many workflow initiatives, the data perspective is often only minimally supported with issues such as persistence, concurrency management and complex data manipulation often being outsourced to third party products. In an effort to characterise the required range of data facilities in a workflow language, *newYAWL* incorporates a series of features derived from the data patterns. These include:

- Support for a variety of *distinct scopes* to which data elements can be bound. This allows the visibility and use of data elements to be restricted. The range of data scopes recognised include: *global* (available to all elements of all process instances), *folder* (available to the elements of process instances to which the folder is currently assigned), *case* (available to all elements in a given process instance), *block* (available to all elements of a specific process or subprocess definition for a given process instance), *scope* (available to a subset of the elements in a specific top-level process or subprocess definition for a given process instance), *task* (available to a given instance of a task) and *multiple-instance* (available to a specific instance of a multiple instance task);
- *Formal parameters* for specifying how data elements are transferred between process constructs (e.g. block to task, composite task to subprocess decomposition, block to multiple instance task). These parameters take a function-based approach to data transfer, thus providing the ability to support inline formatting of data elements and setting of default values. Parameters can be associated with tasks, blocks and processes;
- *Link conditions* for specifying conditions on outgoing arcs from OR-splits and XOR-splits that allow the determination of whether these branches should be activated;
- *Preconditions* and *postconditions* for tasks and processes. They are evaluated at the enablement or completion of the task or process with which they are associated. Unless they evaluate to true, the task or process instance with which they are associated cannot commence or complete execution; and
- *Locks* which allows tasks to specify data elements that they require exclusive access to (within a given process instance) in order to commence. Once these data elements are available, the associated task instance retains a lock on them until it has completed execution preventing any other task instances from using them concurrently. The lock is relinquished once the task instance completes.

2.3 Resource perspective

The resource perspective in *newYAWL* provides a variety of means of controlling and optimising the way in which work is distributed to users and the manner in which it is progressed through to ultimate completion. For each task, a specific *interaction strategy* can be specified which precisely describes the way in which the work item will be communicated to the user, how their commitment to executing it will be established and how the time of its commencement will be determined. Similarly, a detailed *routing strategy* can be defined which determines the range of potential users that can undertake the work item. The routing strategy can nominate the potential users in a variety of ways — they can be directly specified by name, in terms of roles that they perform or the decision as to possible users can be deferred to runtime. There is also provision for determining the range of potential users based on capabilities that individual users possess, the organisational structure in which the process operates or the results of preceding execution history. The routing strategy can be further refined through the use of constraints that restrict the potential user population. Indicative constraints may include: *retain familiar* (i.e. route to a user that undertook a previous work item), *four eyes principle* (i.e. route to a different user than one who undertook a previous work item), *random allocation* (route to a user at random from the range of potential users), *round robin allocation* (route to a

user from the potential population on an equitable basis such that all users receive the same number of work items over time) and *shortest queue allocation* (route the work item to the user with the shortest work queue).

newYAWL also supports two advanced operating modes that are designed to expedite the throughput of work by imposing a defined protocol on the way in which the user interacts with the system and work items are allocated to them. These modes are: *pled execution* where all work items corresponding to a given task are routed to the same user and *chained execution* where subsequent work items in a process instance are routed to the same user once they have completed a preceding work item. Finally, there is also provision for specifying a range of user privileges, both at process and individual task level, that restrict or augment the range of interactions that they can have with the process engine when they are undertaking work items.

3 Mapping *newYAWL* to Coloured Petri Nets

The language design for *newYAWL* is made up of two distinct components: (1) an *abstract syntax* that characterises the various constructs of which the language is comprised and the relationships between them, hence facilitating the capture of a *newYAWL* business process model from static perspective, and (2) a *semantic model* which describes the enactment of a *newYAWL* business process model.

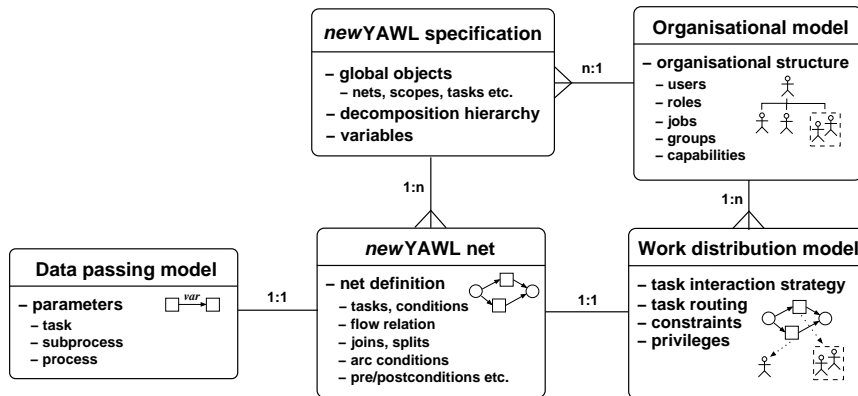


Fig. 3. Schema definition comprising the *newYAWL* abstract syntax

The *newYAWL* abstract syntax is composed of five distinct schemas that capture various aspects of a business process model. Each of these schemas is specified on a set-theoretic basis. Figure 3 summarises the content captured by each of the individual schemas and the relationships between them. Each process captured using the *newYAWL* abstract syntax has a single instance of the *newYAWL* specification associated with it. This defines elements that are common to all of the schemas and also captures the decomposition hierarchy. Each *newYAWL* specification is associated with an instance of the organisational model that describes which users are available to undertake tasks that comprise the process and the organisational context in which they operate.

A *newYAWL* process can be made up of a series of distinct subprocesses (where each subprocess specifies the manner in which a composite task is implemented) together with the top-level process. For each of these (sub)processes, there is an instance of the *newYAWL* net which describes the structure of the (sub)process in detail in terms of the tasks that it comprises and the sequence in which they occur. Associated with each *newYAWL* net is a data passing model which defines the way in which data is passed between elements in the process in terms of formal parameters operating between these elements. There is also a work distribution model that defines how each task will be routed to users for execution, any constraints associated with this activity and privileges that specific users may have assigned to them. The collective group of schemas for a specific process model is termed a *complete newYAWL* specification.

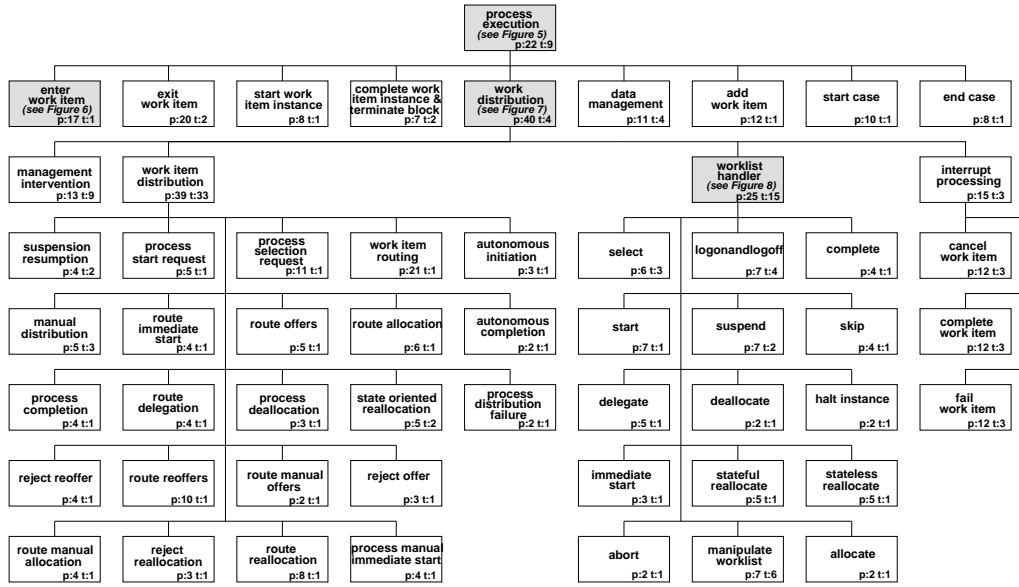


Fig. 4. *newYAWL* CPN model hierarchy (top-level in Figure 4)

The semantics of *newYAWL* are defined in terms of a series of interrelated Coloured Petri Nets developed using the CPN Tools environment. This approach to formalising the language offers the dual benefits of establishing a precise definition of the operation of each of the constructs which comprise *newYAWL* and also providing a means by which an instance of a *newYAWL* specification can be directly executed. There are 55 distinct CPNs which make up the semantic model. These are illustrated in Figure 4 along with the relationships between them. An indication of the complexity of individual nets is illustrated by the p and t values included for each of them which indicate the number of places and transitions that they contain. It is not possible to discuss the operation of all of these nets in the confines of this paper, however several of them (indicated by the shaded boxes and cross-references) are discussed in further detail in subsequent sections. A comprehensive description of the 55 CPNs which comprise the semantic model together with details of how it is initialised in order to facilitate the execution of a given *newYAWL* process model can be found in [RHEA07].

3.1 Overview of the semantic model

The semantic model logically divides into two main parts: (1) the control-flow and data sections and (2) the work distribution, organisational model and resource management sections. These roughly correspond to the *newYAWL* specification, *newYAWL* net and Data passing model, and the Organisational model and Work distribution model illustrated in Figure 3 respectively, which in turn seek to capture the majority of control-flow, data and resource patterns.

Figure 5, which is the topmost net in the semantic model, provides a useful summary of the major components and their interrelationship. The various aspects of control-flow, data management and work distribution information from the static *newYAWL* specification are encoded into the CPN model as tokens in individual places. The top level view of the lifecycle of a process instance is indicated by the transitions in this diagram connected by the thick black line. First a new process instance is started, then there are a succession of **enter**→**start**→**complete**→**exit** transitions which fire as individual task instances are enabled, the work items associated with them are started and completed and the task instances are finalised before triggering subsequent tasks in the process model. Each atomic work item needs to be distributed to a suitable resource for execution, an act which occurs via the **work distribution** transition. This cycle repeats until the last task instance in the process is completed. At this point, the process instance is terminated via the **end case** transition. There is provision for data interchange between the process instance and the environment via the **data management** transition. Finally where a process model supports task concurrency via multiple work item instances, there is provision for the dynamic addition of work items via the **add** transition.

The major data items shared between the activities which facilitate the process execution lifecycle are shown as shared places in this diagram. Not surprisingly, this includes both *static* elements which describe characteristics of individual processes such as the flow relation, task details, variable declarations, parameter mappings, preconditions, postconditions, scope mappings and the hierarchy of processes and subprocesses which make up an overall process model, all of which remain unchanged during the execution of particular instances of the process. It also includes *dynamic* elements which describe how an individual process instance is being enacted at any given time. These elements are commonly known as the *state* of a process instance and include items such as the current marking of the place in the flow relation, variable instances and their associated values, locks which restrict concurrent access to data elements, details of subprocesses currently being enacted, folder mappings (identifying shared data folders assigned to a process instance) and the current execution state of individual work items (e.g. *enabled*, *started* or *completed*).

There is relatively tight coupling between the places and transitions in Figure 5, illustrating the close integration that is necessary between the various aspects of the control-flow and data perspectives in order to enact a process model. The coupling between these places and the **work distribution** transition however is much looser. There are no static aspects of the process that are shared with other transitions in the model (i.e. the transitions underpinning **work distribution**) and other than the places which serve to communicate work items being distributed to resources for execution (and being started, completed or cancelled), the **variable instances** place is the only aspect of dynamic data that is shared with the work distribution

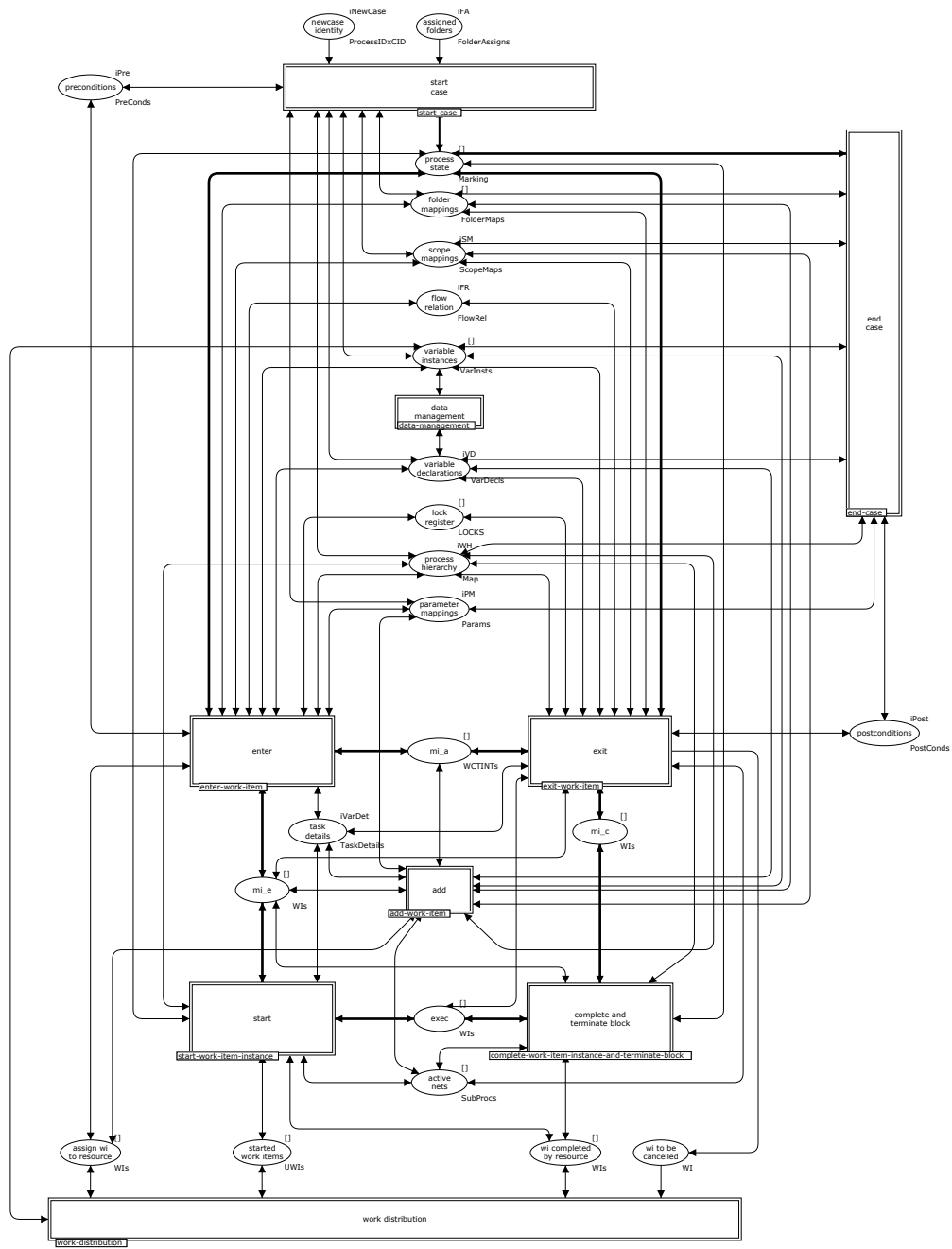


Fig. 5. Overview of the *newYAWL* semantic model

subprocess. The following sections focus on the two main parts of the *newYAWL* semantic model: (1) control-flow and data handling and (2) work distribution.

3.2 Control-flow and data handling

The actions comprising the control-flow and data handling processes are extremely complex both in terms of the range of concepts that they involve and the interrelationship between them. It is not possible to describe all aspects of these processes in the confines of this paper hence in this section we focus on one specific aspect of the overall work item lifecycle: *task instance enablement* and *work item creation*. Task instance enablement is the first step in work item execution. It is depicted by the **enter** transition in Figure 6. The first step in determining whether a task instance can be enabled is to examine the marking of the input places to the task. There are four possible scenarios:

- If the task has no joins associated with it, then the input condition to the task simply needs to contain a token;
- If the task has an AND-join associated with it, each input condition needs to contain a token with the same *ProcessID* × *CID* combination, where these two attributes uniquely identify a process and process instance (or case) respectively;
- If the task has an XOR-join associated with it, one of the input conditions needs to contain a token; and
- If the task has an OR-join associated with it, one (or more) of the input conditions needs to contain a token and a determination needs to be made as to whether in any future possible state of the process instance, the currently marked input conditions can retain at least one token and another input condition can also receive a token. If this can occur, the task is not enabled, otherwise it is enabled. This issue has been subject to rigorous analysis and an algorithm has been proposed [WEAH05] for determining exactly when an OR-join can fire. The *newYAWL* semantic model implements this algorithm.

Depending on the form of task that is being enabled (singular or multiple-instance), one or more work items may be created for it. If the task is atomic, the work item(s) is created in the same block as the task to which it corresponds. If the task is composite, then the situation is slightly more complicated and two things occur: (1) a “virtual” work item is created in the same block for each instance of the task that will be initiated (this enables later determination of whether the composite task is in progress or has completed) and (2) a new subprocess decomposition (or a new block) is started for each task instance. This involves the placement of a token in the input place to the subprocess decomposition which has a distinct subprocess CID. Table 1 indicates the potential range of work items that may be created for a given task instance. In order for a task to be enabled, all prerequisites associated with the task must be satisfied. There are five prerequisites for the **enter** transition to be able to fire:

- The precondition associated with the task must evaluate to true;
- All data elements which are inputs to mandatory input parameters must exist and have a defined value;
- All mandatory input parameters must evaluate to defined values;

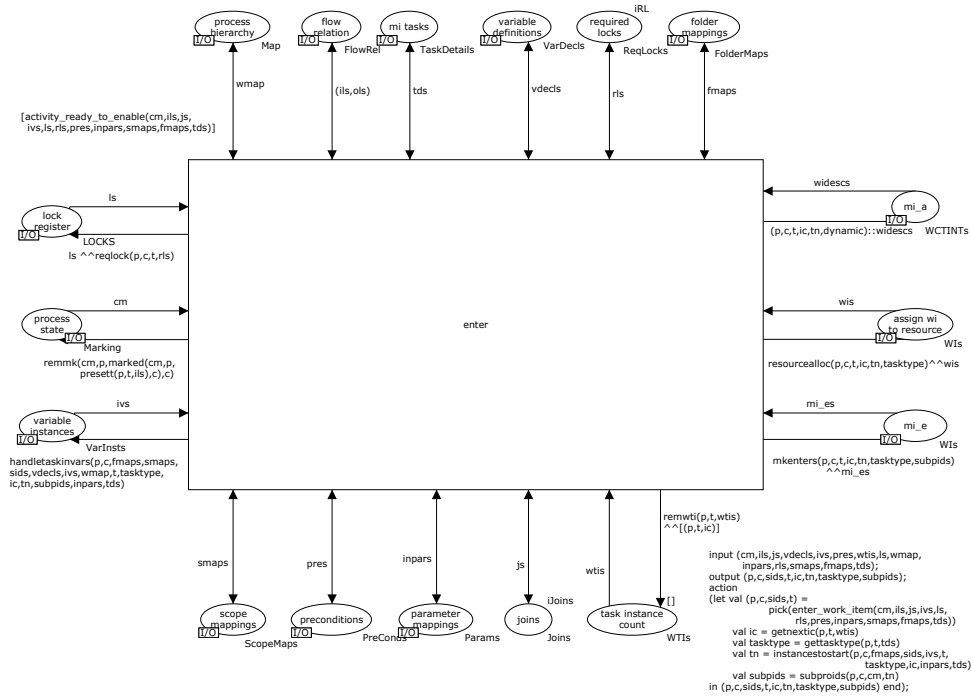


Fig. 6. Enter work item transition

- All locks which are required for data elements that will be used by the work items associated with the task must be available; and
- If the task is a multiple instance task, the multiple instance parameter when evaluated must yield a number of rows that is between the minimum and maximum number of instances required for the task to be initiated.

Once these prerequisites are satisfied, task enablement can occur. This involves:

1. Removing the tokens marking input conditions to the task corresponding to the instance enabled. The exact number of tokens removed depends on whether there is a join associated with the task or not and occurs as follows:
 - *No join*: one token corresponding to the *ProcessID* × *CID* combination that triggered the task is removed from the input condition to the task;
 - *AND-join*: one token corresponding to the triggering *ProcessID* × *CID* combination is removed from each of the input conditions to the task;
 - *XOR-join*: one token corresponding to the triggering *ProcessID* × *CID* combination is removed from *one* of the input conditions to the task; and
 - *OR-join*: one token corresponding to the triggering *ProcessID* × *CID* combination is removed from any of the input conditions to the task which currently contain tokens of this form.
2. Determining which instance of the task this is. The instance identifier (which corresponds to the *Inst* attribute) must be unique for each task instance and all work items and data elements associated with this task instance in order to

ensure that they can be uniquely identified. A record is kept of the next available instance for a task in the `task instance count` place.

3. Determining how many work item instances should be created. For a singular task (i.e. an atomic or composite task), this will always be a single work item, however for a multiple instance task (i.e. an atomic or composite multiple instance task), the actual number started will be determined from the evaluation of the multiple instance parameter which will return a composite result containing a number of rows of data. The number of rows returned indicates the number of instances to be started. In all of these situations, individual work items are created which share the same *ProcessID*, *CID*, *TaskID* and *Inst* values, however the *TaskNr* value is unique for each work item and is in the range *1..number of work items created*;
4. For all work items corresponding to composite tasks, distinct subprocess CIDs need to be determined to ensure that any variables created for subprocesses are correctly identified and can be accessed by the work items for the subprocesses that will subsequently be triggered;
5. Creating variable instances for data elements associated with the task. This varies depending on the task type and the number of work items created for the task:
 - For atomic tasks which only have a single instance, this will involve the creation of relevant task variables.
 - For atomic multiple instance tasks, this will involve the creation of both task variables and multiple instance variables for each task instance. The required multiple instance variables are indicated by the output data elements listed for the multiple instance parameter. and this set of variables is created for each new work item.
 - For composite tasks that only have a single instance, any required task variables are created in the subprocess decomposition that is instantiated for the task. Also, there may be block and scope variables associated with the subprocess decomposition that need to be created; and
 - For composite multiple instance tasks, any required block, scope, task variables and multiple instance variables are created for each subprocess decomposition that is initiated for the task.
6. Mapping the results of any input parameters for the task instance to the relevant output data elements. For multiple instance parameters, this can be quite a complex activity;
7. Recording any variable locks that are required for the execution of the task instance;
8. For all work items corresponding to atomic tasks (other than for automatic tasks which can be initiated without distribution to a resource), requests for work item distribution need to be created. These are routed to the `assign wi to resource` place and are subsequently dealt with by the `work distribution` transition; and
9. Finally, work items with an *enabled* status need to be created for this task instance and added to the `mi_e` place (which identifies work items corresponding to enabled but not yet started tasks) in accordance with the details outlined in Table 1.

Table 1. Task instance enablement in *newYAWL*

Task Type	Instances Initiated at Commencement	
	<i>Singular</i>	<i>Multiple Instances</i>
<i>Atomic</i>	Single work item created in the same block.	Multiple work items created in the same block, each with a distinct <i>TaskNr.</i>
<i>Composite</i>	Single “virtual” work item created in the same block and a new subprocess is initiated for the block assigned as the task decomposition.	Multiple “virtual” work items created in the same block. Additionally a distinct subprocess is initiated for each work item created, each with a distinct subprocess <i>CID</i> and <i>TaskNr.</i>

The work distribution process in *newYAWL* provides an interesting contrast to the control-flow and data handling. Unlike the latter process which is comprised of a limited number of transitions which must coordinate state changes involving a large number of places with complex guard conditions, the work distribution process is much more diffuse. It involves multiple places which describe alternate states for a work item that is currently in progress and supports a variety of distinct transitions between these states. The work distribution process is discussed in the next section.

3.3 Work distribution

The main motivation for workflow systems is achieving more effective and controlled distribution of work. Hence the actual distribution and management of work items are of particular importance. The process of distributing work items is summarized by Figure 7. It comprises four main components⁴:

- the **work item distribution** transition, which handles the overall management of work items through the distribution and execution process;
- the **worklist handler**, which corresponds to the user-facing client software that advises users of work items requiring execution and manages their interactions with the main **work item distribution** transition in regard to committing to execute specific work items, starting and completing them;
- the **management intervention** transition, that provides the ability for a process administrator to intervene in the **work distribution** process and manually reassign work items to users where required; and
- the **interrupt handler** transition that supports the cancellation, forced completion and forced failure of work items as may be triggered by other components of the process engine (e.g. the control-flow process, exception handlers).

Work items that are to be distributed through this process are added to the **work items for distribution** place. This then prompts the **work item distribution** transition to determine how they should be routed for execution. This may involve the services of the process administrator in which case they are sent to the **management intervention** transition or alternatively they may be sent directly to one or

⁴ Note that the high-level structure of the work distribution process is influenced by the earlier work of Pesic and van der Aalst [PA07].

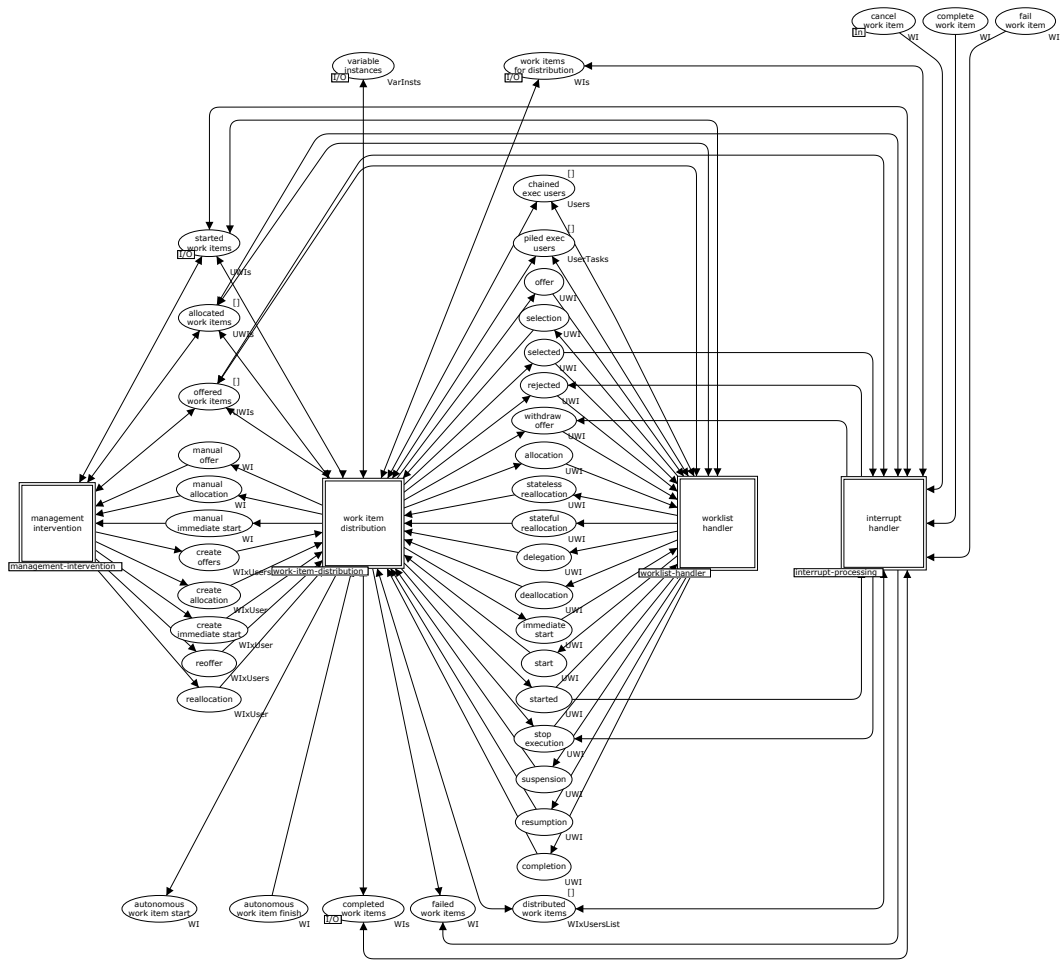


Fig. 7. Top level view of the main *work distribution* process

more users via the **worklist handler** transition. The various places between these three activities correspond to the range of requests that flow between them. In the situation where a work item corresponds to an *automatic* task, it is sent directly to the **autonomous work item start** place and no further distribution activities take place. An automatic task is considered complete when a token is inserted in the **autonomous work item finish** place.

A common view of work items in progress is maintained between the **work item distribution**, **worklist handler** and **management intervention** transitions via the **offered work items**, **allocated work items** and **started work items** places. There is also shared information about users in advanced operating modes that is recorded in the **piled exec users** and **chained exec users** places. Although there is significant provision for shared information about the state of work items, the determination of when a work item is actually complete rests with the **work item distribution** transition and when this occurs, it inserts a token in the **completed**

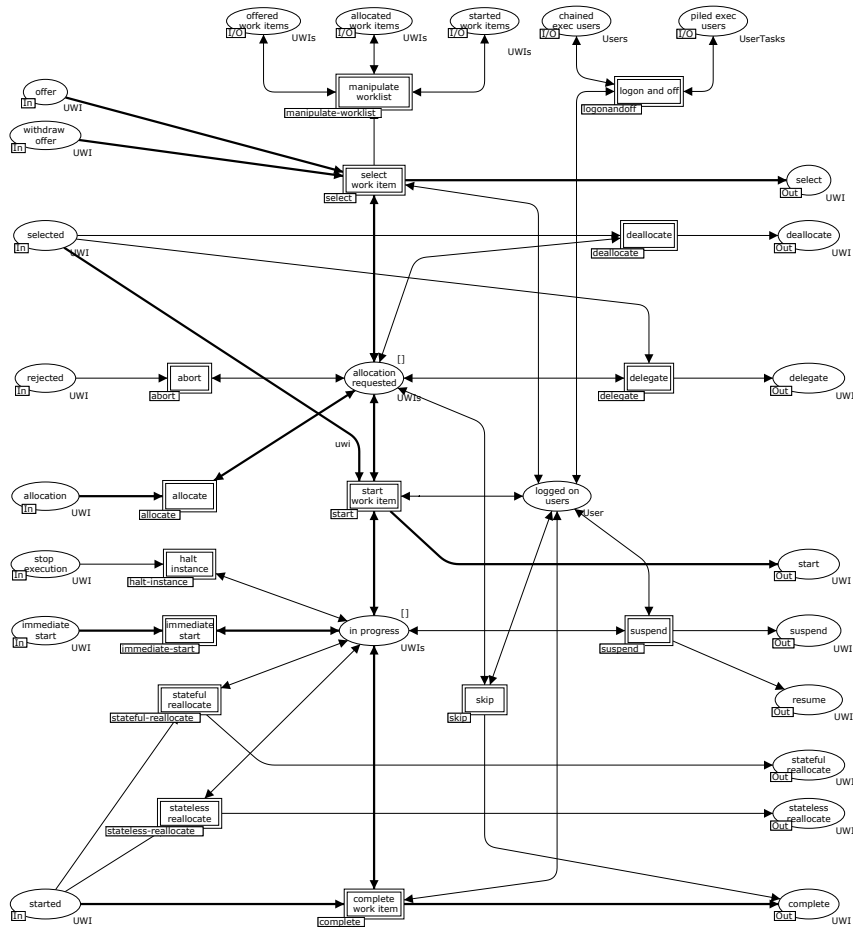


Fig. 8. Worklist handler process

work items place. Similarly, work item failures are notified via the failed work items place. The only exception to these arrangements are for work items that are subject to some form of interrupt (e.g. an exception being detected and handled). The interrupt handler transition is responsible for managing these occurrences on the basis of cancellation, forced completion and failure requests received in the cancel work item, complete work item and fail work item places respectively. All of the activities in the work distribution process are illustrated by substitution transitions indicating that each of them are defined in terms of significantly more complex subprocesses. It is not possible to present each of them in this paper, hence we focus on one of the more significant: the worklist handler process. The worklist handler is illustrated in Figure 8 and describes how the user-facing process interface (typically a worklist handler software client) operates and interacts with the work item distribution process. As previously, the main path through this process is indicated by the thick black arcs. There are various transitions that make up the process, these correspond to actions that individual users can request

in order to alter the current state of a work item to more closely reflect their current handling of it. These actions may simply be requests to start or complete it or they may be “detour” requests to reroute it to other users e.g. via **delegation** or **deallocation**. The manner in which these requests operate is illustrated by the shared places in Figure 7.

4 Related work

There have been numerous papers advocating approaches to workflow and business process modelling based on Petri Nets (cf. [Aal98],[EN93],[AAH98],[MR03]), however these tend to either focus on a single aspect of the domain (e.g. the control-flow perspective) or they are based on a relatively simplistic language. There have also been attempts to provide formal semantics using Petri Nets for many of the more widely used approaches to business process modelling including EPCs [Aal99], UML 2.0 Activity Diagrams [SH05] and BPMN [DDO07], although in each case arriving at a complete semantics has been hampered by inherent ambiguities in the informal descriptions for each of the formalisms. There has been minimal work on formalisation of the other workflow perspectives, one exception is [PA07] which investigates mechanisms for work distribution in workflows and presents CPN models for a number of the workflow resource patterns.

Historically, the modelling and enactment of processes have often been treated distinctly and it is not unusual for separate design and runtime models to be utilised by systems. Approaches to managing the potential disparities between these models have included the derivation of executable process descriptions from design-time models [DNLS⁺02] and the direct animation of design-time models for requirements validation [MLO⁺07]. The latter of these approaches which uses a strategy based on Coloured Petri Nets [Jen97] and CPN Tools [JKW07] as an enablement vehicle is one of a number of initiatives that have successfully used the CPN Tools offering as a means of executing various design-time modelling formalisms including Protos models [GAJVV06], Sequence diagrams [RF06] and task descriptions [JLA06].

5 Experiences and conclusions

The *newYAWL* semantic model⁵ incorporates 55 distinct pages of CPN diagrams and encompasses 480 places, 138 transitions and in excess of 1500 lines of ML code. It took approximately six months to develop. The size of the model gives an indication of the relative complexity of formally specifying a comprehensive business process modelling language such as *newYAWL*. Indeed, it is only with the aid of an interactive modelling environment such as CPN Tools that developing a formalisation of this scale actually becomes viable. One of the major advantages of pursuing this approach to software development is that it provides a design that is executable. This allows fundamental design decisions to be evaluated and tested much earlier than would ordinarily be the case during the development process. Where suboptimal design decisions are revealed, the cost of rectifying them is significantly less than it would be later in the development lifecycle. There is also the opportunity to test alternate solutions to design issues with minimal overhead before a final decision is

⁵ This model is available at www.yawl-system.com/newYAWL.

settled on. A particular benefit afforded by this approach to formalisation is that the CPN hierarchy established during the design process provides an excellent basis on which to make subsequent architectural and development decisions.

The original motivations for this research initiative were twofold: (1) to establish a fully formalised business process modelling language based on the synthesis of the workflow patterns and (2) to demonstrate that the language was not only suitable for conceptual modelling of business processes but that it also contained sufficient detail for candidate models to be directly enacted. *newYAWL* achieves both of these objectives and directly supports 118 of the 126 workflow patterns that have been identified. It is interesting to note however that whilst the development of a model of this scale offers some extremely beneficial insights into the overall problem domain and provides a software design that can be readily utilised as the basis for subsequent programming activities, it also has its limitations. Perhaps the most significant of these is that the scale and complexity of the model obviates any serious attempts at verification. Even on a relatively capable machine (P4 1.6Ghz, 512Mb RAM), the model takes over 8 minutes to load. Moreover the potentially infinite range of business process models that the semantic model can encode, rules out the use of techniques such as state space analysis. This raises the question as to how models of this scale can be comprehensively tested and verified.

Notwithstanding these considerations however, the development of the semantic model delivered some salient insights into areas of newYAWL that needed further consideration during the formalisation activity. These included:

- recognition of the fact that at runtime the completion region construct can only bring affected work items to the point at which they *should* complete. It cannot force the completion to occur;
- recognition that when a self-cancelling task completes: (1) it should process the cancellation of itself last of all in order to prevent the situation where it cancels itself before all other cancellations have been completed and (2) it needs to establish whether it is cancelling itself before it can make the decision to put tokens in any relevant output places associated with the task;
- introduction of a consistent approach for handling the evaluation of any functions associated with a *newYAWL* specification e.g. for outgoing links in a XOR-split, pre/postconditions, pre/post tests for iterative tasks etc. This issue was ultimately addressed by mapping any necessary function calls to ML functions and establishing a standard approach to encoding the invocation of these functions and the passing of any necessary parameters and the return of associated results;
- adoption of a standard strategy for characterising parameters to functions in order to ensure that they could be passed in a uniform way to the associated ML functions that evaluated them;
- the establishment of a coherence protocol to ensure that reallocation of work items to alternate resources either by users or the process administrator are handled in a consistent manner in order to ensure that potential race conditions arising during reallocation do not result in the process engine, process administrator or the initiating user having differing views of the current state of work item allocations; and
- recognition that the current approach to privilege specification for users and tasks (where privileges need to be individually specified) is likely to be intractable for any large scale implementation of *newYAWL*.

There were also some learnings in regard to the CPN Tools offering during the course of this research. Whilst extremely powerful, there are several aspects of the CPN Tools environment that would benefit from the inclusion of additional capabilities. In particular, the ability to incrementally wind back the execution state of a given execution would be useful, as would the ability to save an execution state for later execution and analysis. The interaction facilities for the CPN model are particularly effective, however there are less features provided for tracing and altering the execution of ML code segments that form part of a CPN model. The inclusion of features such as these in future CPN Tools releases would be extremely beneficial.

The *new*YAWL semantic model will serve as the design blueprint for the next major version of the open-source YAWL System offering. This is currently being developed by the BPM Group at QUT.

References

- [AAH98] N.R. Adam, V. Atluri, and W.K. Huang. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information Systems*, 10(2):131–158, 1998.
- [Aal98] W.M.P. van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [Aal99] W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.
- [AH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.
- [AHKB03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.
- [DDO07] R.M. Dijkman, M. Dumas, and C. Ouyang. Formal semantics and automated analysis of BPMN process models. Technical Report 5969, Queensland University of Technology, Brisbane, Australia, 2007. <http://eprints.qut.edu.au/archive/00005969/>.
- [DNLS⁺02] E. Di Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta. Deriving executable process descriptions from UML. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 155–165, New York, NY, USA, 2002. ACM Press.
- [EN93] C.A. Ellis and G.J. Nutt. Modelling and enactment of workflow systems. In M. Ajmone Marsan, editor, *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16, Chicago, IL, USA, 1993. Springer.
- [GAJVV06] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and H.M.V. Verbeek. Protos2CPN: Using colored Petri nets for configuring and testing business processes. In K. Jensen, editor, *Proceedings of the 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-579 of *Daimi Reports*, pages 137–155, Aarhus, Denmark, 2006.
- [Jen97] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1997.
- [JKW07] K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *International Journal of Software Tools for Technology Transfer*, 9(3):213–254, 2007.
- [JLA06] J.B. Jørgensen, K.B. Lassen, and W.M.P. van der Aalst. From task descriptions via coloured Petri nets towards an implementation of a new electronic patient

- record. In K. Jensen, editor, *Proceedings of the 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-579 of *Daimi Reports*, pages 137–155, Aarhus, Denmark, 2006.
- [MLO⁺07] R.J. Machado, K.B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Requirements validation: Execution of UML models with CPN Tools. *International Journal on Software Tools for Technology Transfer*, 9(3):353–369, 2007.
- [MR03] Daniel Moldt and Heiko Rölke. Pattern based workflow design using Reference nets. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *Proceedings of the Business Process Management Conference 2003*, volume 2678 of *Lecture Notes in Computer Science*, pages 246–260, Eindhoven, The Netherlands, 2003. Springer.
- [PA07] M. Pesic and W.M.P. van der Aalst. Modelling work distribution mechanisms using colored Petri nets. *International Journal on Software Tools for Technology Transfer*, 9(3):327–352, 2007.
- [RAHE05] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In O. Pastor and J. Falcão e Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232, Porto, Portugal, 2005. Springer.
- [RF06] O.R. Ribeiro and J.M. Fernandes. Some rules to transform sequence diagrams into coloured Petri nets. In K. Jensen, editor, *Proceedings of the 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-579 of *Daimi Reports*, pages 137–155, Aarhus, Denmark, 2006.
- [RHAM06] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. Technical Report BPM-06-22, 2006. <http://www.BPMcenter.org>.
- [RHEA05] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In L. Delcambre, C. Kop, H.C. Mayr, J. Mylopoulos, and O. Pastor, editors, *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, volume 3716 of *Lecture Notes in Computer Science*, pages 353–368, Klagenfurt, Austria, 2005. Springer.
- [RHEA07] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. newYAWL: achieving comprehensive patterns support in workflow for the control-flow, data and resource perspectives. Technical Report BPM-07-05, 2007. <http://www.BPMcenter.org>.
- [RZ96] D. Riehle and H. Züllighoven. Understanding and using patterns in software development. *Theory and Practice of Object Systems*, 2(1):3–13, 1996.
- [SH05] H. Störrle and J.H. Hausmann. Towards a formal semantics of UML 2.0 activities. In P. Liggesmeyer, K. Pohl, and M. Goedicke, editors, *Proceedings of the Software Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik*, volume 64 of *Lecture Notes in Informatics*, pages 117–128, Essen, Germany, 2005. Gesellschaft für Informatik.
- [WEAH05] M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a general, formal and decidable approach to the OR-join in workflow using Reset nets. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International Conference on Application and Theory of Petri nets and Other Models of Concurrency (Petri Nets 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443, Miami, USA, 2005. Springer-Verlag.
- [Wor95] Workflow Management Coalition. Reference model — the workflow reference model. Technical Report WFMC-TC-1003, 19-Jan-95, 1.1, 1995. <http://www.wfmc.org/standards/docs/tc003v11.pdf>.