*new*YAWL: Designing a Workflow System using Coloured Petri Nets[⋆⋆⋆]

Nick Russell[1], Wil M.P. van der Aalst[1,2] and Arthur H.M. ter Hofstede[2]

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

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

Abstract. Traditional workflow systems focus on providing support for the control-flow perspective of a business process, with other aspects such as data management and work distribution receiving markedly less attention. A guide to desirable workflow characteristics is provided by the well-known workflow patterns which are derived from a comprehensive survey of contemporary tools and modelling formalisms. In this paper we describe the approach taken to designing the *new*YAWL workflow system, an offering that aims to provide comprehensive support for the control-flow, data and resource perspectives based on the workflow patterns. The semantics of the *new*YAWL workflow language are based on Coloured Petri Nets thus facilitating the direct enactment and analysis of processes described in terms of *new*YAWL language constructs. As part of this discussion, we explain how the operational semantics for each of the language elements are embodied in the *new*YAWL system and indicate the facilities required to support them in an operational environment. We also review the experiences associated with developing a complete operational design for an offering of this scale using formal techniques.

1 Introduction

There are a plethora of workflow systems on the market today providing organisations with various forms of automated support for their business processes. It is ironic however, that despite the rigour that workflow systems introduce into the conduct of the processes that they coordinate, they themselves do not demonstrate the same rigour in the workflow languages that they enact. Indeed, it is a salient fact that, almost without exception, workflow languages are defined

[⋆] An earlier version of this work was presented at CPN'07, the Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools [21].

[⋆⋆] 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).

on an informal basis leaving their precise operation unclear to anyone other than the system developers. An additional shortcoming of existing workflow solutions is their focus on the control-flow aspects of business processes.

The *YAWL* Initiative sought to address the first of these issues. *YAWL* [3] is an acronym for *Yet Another Workflow Language*. It provides 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 [4]. 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*[1] 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 alike[2].

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 aspects of processes, however this effort was hampered by the lack of a complete formal description of the requirements in these perspectives. Recent work conducted as part of the Workflow Patterns Initiative has identified the core elements in other process perspectives (data, resource, exception handling) and a recent review [22] of the control-flow perspective has 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 — termed *new*YAWL has been proposed — which aims to support the broadest range of the workflow patterns in the control-flow, data and resource perspectives. *new*YAWL synthesises this work to provide a fully formalised workflow language based on a comprehensive view of a business process. The validation of this proposal is to design (and ultimately build) the workflow system that embodies the workflow language. An interesting consequence of formalising the operational semantics for the language constructs in *new*YAWL, has been the establishment of the functional architecture for the system to be developed. This is based on a detailed consideration of the causal effects and data interactions required to support each of the language constructs and their behaviour in a broader operational environment. This paper outlines the approach

[1] See `http://www.yawl-system.com` for further details of the YAWL System and to download the latest version of the software.

[2] Hereafter in this paper, we refer to the collective group of YAWL offerings developed to date — both the YAWL language as defined in [3] and also more recent *YAWL System* implementations of the language based on the original definition (up to and including release Beta 8.2) — as *YAWL*.

taken to designing the *new*YAWL system. In this paper we not only describe the design of *new*YAWL using Coloured Petri Nets, but also reflect on the use of such a design approach from a software engineering standpoint.

The remainder of this paper proceeds as follows: Section 2 introduces the language constructs which comprise *new*YAWL. Section 3 describes the approach to designing a workflow system that can enact business processes described in terms of the *new*YAWL language. Section 4 overviews related work and Section 5 discusses the experiences of designing a workflow system using formal methods and concludes the paper.

2 *new*YAWL: the language

*new*YAWL 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 together with an executable semantic model in the form of Coloured Petri Nets which define the runtime semantics of each of the language constructs. The following sections provide an overview of the features of *new*YAWL 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 *new*YAWL. 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 [3]. Several new constructs have been added based on the full range of workflow patterns that have now been identified. These are:

- 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

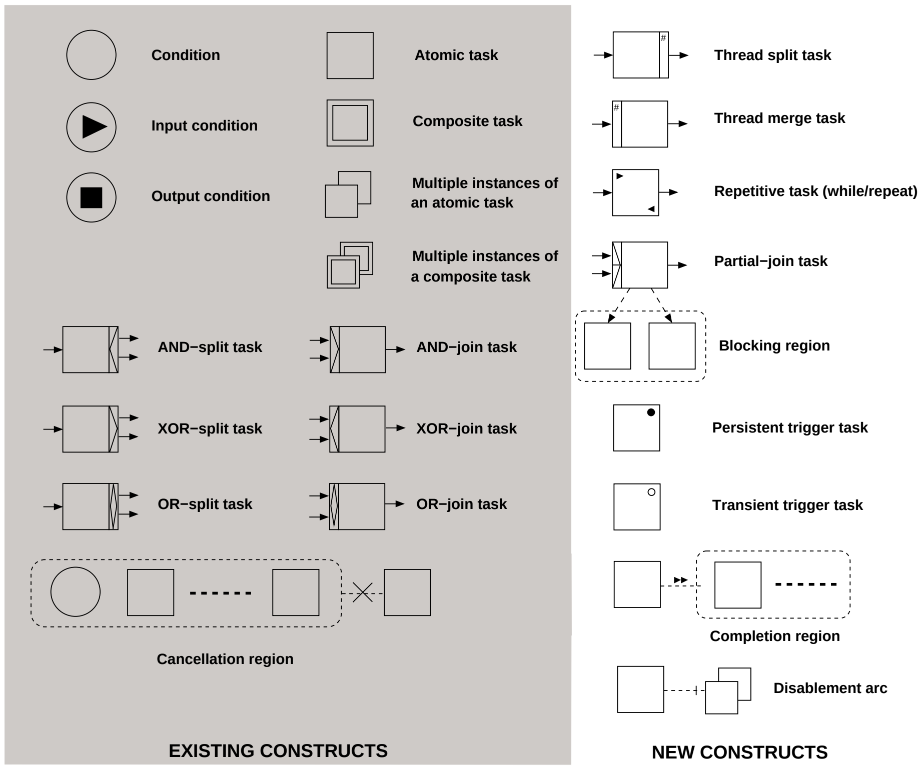

Fig. 1. *new*YAWL symbology

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 loop);

– 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

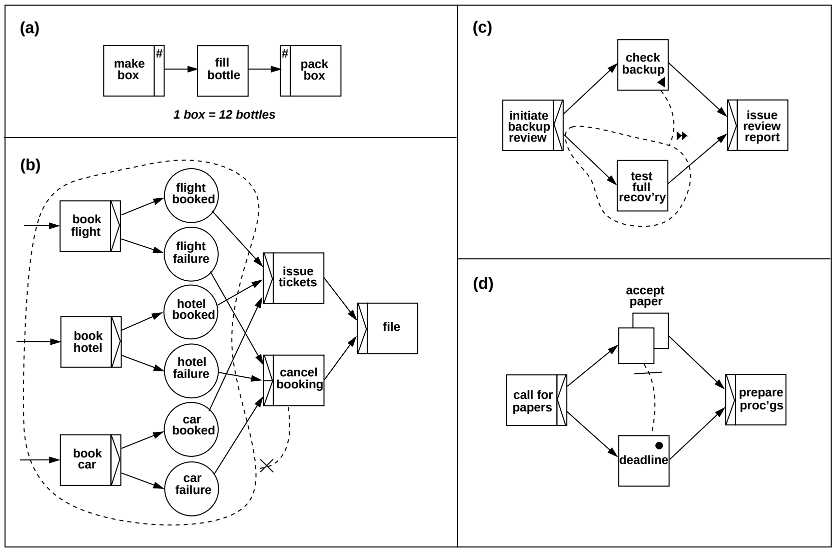

Fig. 2. Examples of *new*YAWL control-flow constructs

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, *new*YAWL 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 allow 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 *new*YAWL 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 recorded 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 a similar number of work items over time) and *shortest queue allocation* (route the work item to the user with the shortest work queue).

*new*YAWL 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: *piled 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 workflow engine when they are undertaking work items.

3 *new*YAWL: the system

A workflow system encompasses a number of distinct functions as illustrated by the diagram in Figure 3. Generally the business process that is to be automated is captured in the form of a *process model*. A *workflow management system* is responsible for coordinating the execution of instances of the process model. It comprises a number of discrete components. First the *workflow engine* is responsible for managing the control-flow and data elements that are associated with each process instance. As the thread of control flows through a process, it results in the triggering of individual tasks that make up the process model. The enabling of a task results in the creation of a new work item which needs to be executed by a human resource. However, in order for this to occur, the identity of one or more suitable resources needs to be determined. This activity is the responsibility of the *work item routing* component and is based on the interpretation of task routing information associated with each task in the context of the current state of the process instance. Once a set of suitable resources have been determined for a work item, it is necessary to advise them of the pending work item. This function is undertaken by the *worklist management* component which places the work item on the worklist of each resource to whom it is to be routed. Thus the workflow management system retains a centralised view of the state of all work items and also provides *workflow administration* facilities should it be necessary to intervene in the normal conduct of this process.

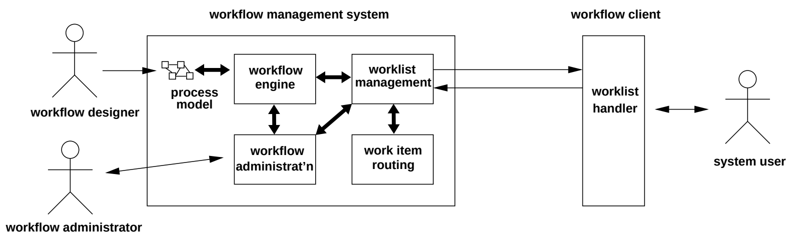

Fig. 3. Outline of major workflow system components

However despite the consistent centralised view of pending work maintained by the workflow management system, there is another layer of complexity in managing the actual distribution and conduct of work items across the range of resources coordinated by the workflow system. This stems from the fact that resources typically operate independently of the workflow system. They retain a distinct view of the work that they are conducting which is accessed via a *worklist handler* (which typically takes the form of a software client running at a distinct location to that of the workflow management system). Although the worklist handler operates on a client-server basis with respect to the workflow management system, it is generally disconnected from it and only connects to it when it wishes to refresh its view of the current work allocation or to advise the workflow system of a change of state in the work items it has been allocated.

Clearly a workflow system involves a relatively complex set of software components and interactions, and in order to provide a precise definition of how a business process should actually be enacted in an operational environment, it is necessary not only to provide an operational semantics for the workflow language that describes the business process, but also to define the overall architecture and operation of the workflow system. This has been done for *new*YAWL using a series of interrelated Coloured Petri Nets developed using the CPN Tools environment [13]. This approach to formalising the system offers the dual benefits of establishing a precise definition of the operation of each of the language constructs which comprise *new*YAWL and also providing a means of describing exactly how an instance of a *new*YAWL specification should be executed. There are 55 distinct CPNs which make up the *new*YAWL system description.

These are illustrated in Figure 4 along with the relationships between them. The correspondence between the functional workflow system components identified in Figure 3 and each of the CPNs is also delineated. 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. Clearly it is not possible to discuss the operation of all of these nets in the confines of this paper, however some 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 *new*YAWL system can be found in [23]. In the following sections, we will outline the operation of three of these areas, illustrated by the shaded models in Figure 4. These provide an overview of the *workflow engine*, *worklist management* and *worklist handler* components of the workflow system.

3.1 Workflow engine

Figure 5, which is the topmost net in the *new*YAWL model, provides a useful summary of the operation of a workflow engine. The various aspects of control-flow, data management and work distribution information which make up a static *new*YAWL specification are encoded in 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

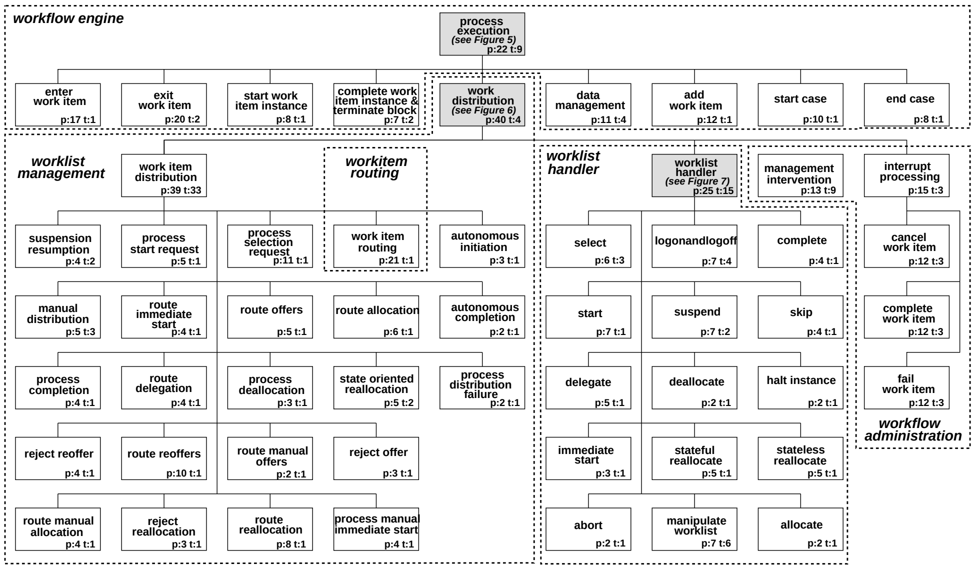
workflow engine
process execution (see Figure 5) p:22 t:9
enter work item p:17 t:1
exit work item p:20 t:2
start work item instance p:8 t:1
complete work item instance & terminate block p:7 t:2
work distribution (see Figure 6) p:40 t:4
data management p:11 t:4
add work item p:12 t:1
start case p:10 t:1
end case p:8 t:1
worklist management
work item distribution p:39 t:33
workitem routing
worklist handler
worklist handler (see Figure 7) p:25 t:15
management intervention p:13 t:9
interrupt processing p:15 t:3
suspension resumption p:4 t:2
process start request p:5 t:1
process selection request p:11 t:1
work item routing p:21 t:1
autonomous initiation p:3 t:1
select p:6 t:3
logonandlogoff p:7 t:4
complete p:4 t:1
cancel work item p:12 t:3
manual distribution p:5 t:3
route immediate start p:4 t:1
route offers p:5 t:1
route allocation p:6 t:1
autonomous completion p:2 t:1
start p:7 t:1
suspend p:7 t:2
skip p:4 t:1
complete work item p:12 t:3
process completion p:4 t:1
route delegation p:4 t:1
process deallocation p:3 t:1
state oriented reallocation p:5 t:2
process distribution failure p:2 t:1
delegate p:5 t:1
deallocate p:2 t:1
halt instance p:2 t:1
fail work item p:12 t:3
reject reoffer p:4 t:1
route reoffers p:10 t:1
route manual offers p:2 t:1
reject offer p:3 t:1
immediate start p:3 t:1
stateful reallocate p:5 t:1
stateless reallocate p:5 t:1
workflow administration
route manual allocation p:4 t:1
reject reallocation p:3 t:1
route reallocation p:8 t:1
process manual immediate start p:4 t:1
abort p:2 t:1
manipulate worklist p:7 t:6
allocate p:2 t:1

Fig. 4. *new*YAWL system CPN model hierarchy

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 routed 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 process and subprocess definitions 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, de-

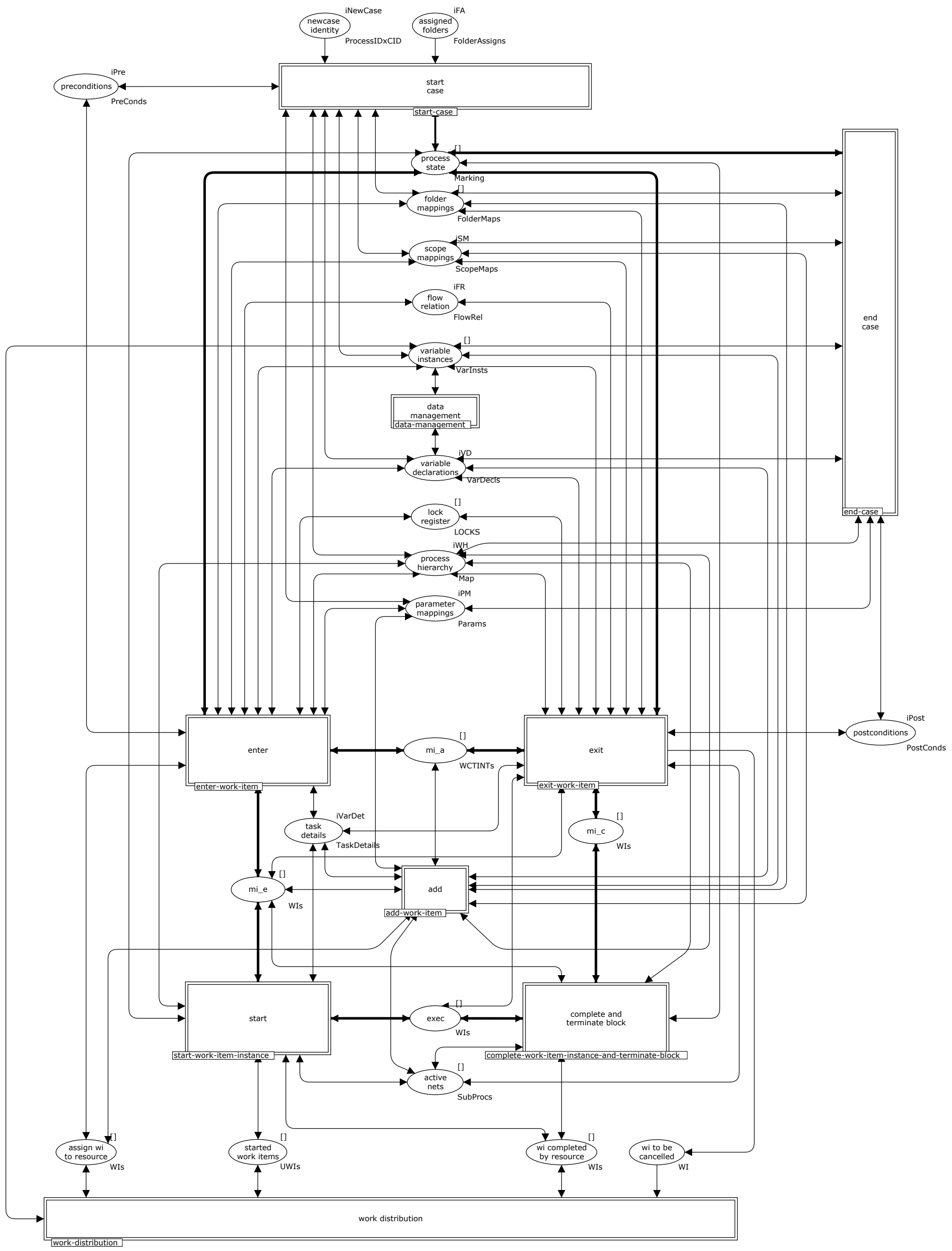

Fig. 5. Overview of the *new*YAWL *workflow engine*

tails 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 subprocess. This reflects the functional independence of the *workflow engine*, *work item routing* and *worklist management* components. The next section looks at the issue of worklist management in more detail.

3.2 Worklist management

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 managing the distribution of work items to resources is summarized by Figure 6. It coordinates the interaction between the *workflow engine*, *work item routing*, *worklist handler* and *workflow administration* components.

The correspondences between these components and the transitions in Figure 6 can be summarised as follows:

- the *worklist management* component is facilitated by the the `work item distribution` transition, which handles the overall management of work items through the distribution and execution process (note that it subsumes the *work item routing* component);
- the *worklist handler* component corresponds directly to the the `worklist handler` transition, which is 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 *workflow administration* component is facilitated via two distinct transitions: the `management intervention` transition, that provides the ability for a workflow 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 workflow engine (e.g. the control-flow process, exception handlers).

Work items that are to be distributed are communicated between the *workflow engine* and the *worklist management* components via the `work items for`

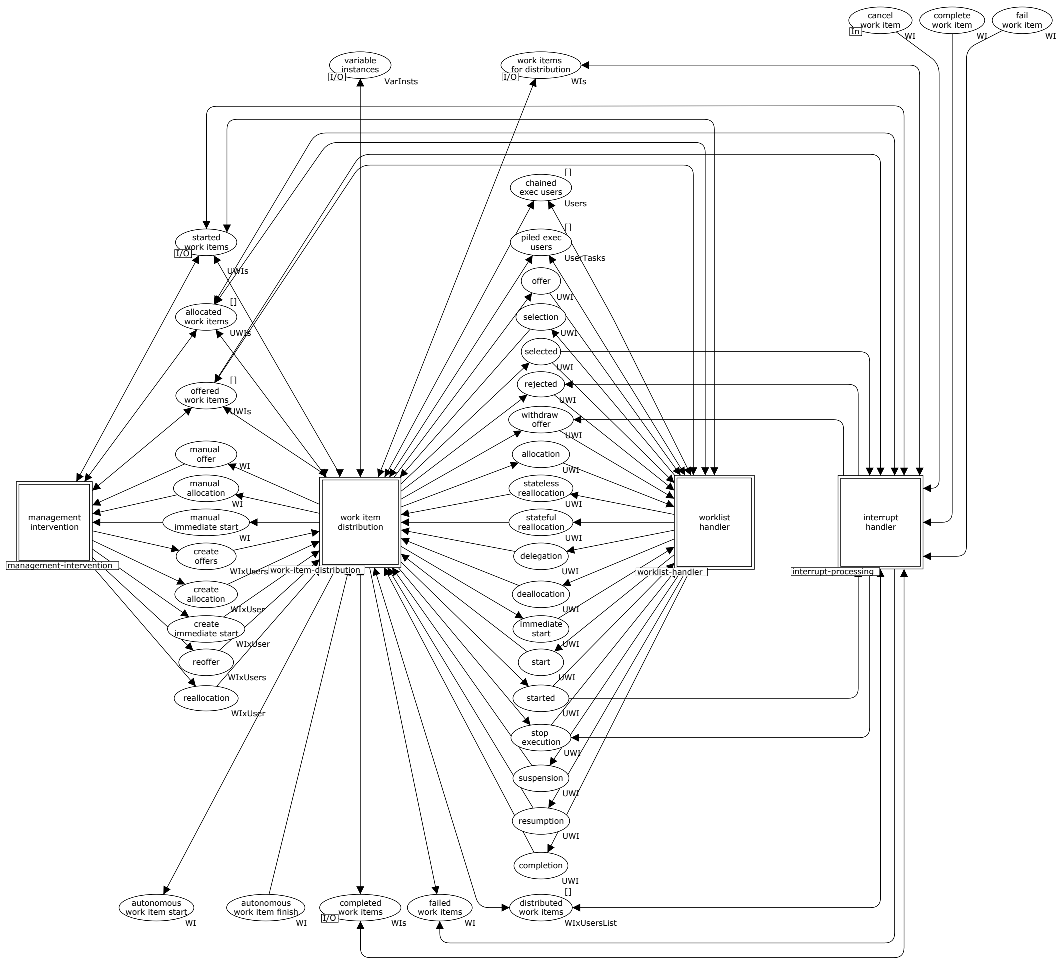

Fig. 6. Top level view of the *worklist management* component

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 workflow administrator in which case they are sent to the `management intervention` transition or alternatively they may be sent directly to one or more resources via the `worklist handler` transition. The various places between these three transitions 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`, `management intervention` and `interrupt` `handler` transitions via the `offered work items`, `allocated work items` and `started work items` places (although obviously this information is only available to the *worklist handler* when it is actually connected to the workflow management system). 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 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 *worklist management* component 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. Finally we focus on one other significant component: the *worklist handler*.

3.3 Worklist handler

The *worklist handler* component is illustrated in Figure 7 and describes how the user-facing workflow interface (typically a worklist handler software client) operates and interacts with the *worklist management* component. 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 6. Typically the inclusion of request in one of these shared places results in a message flowing between the *worklist handler* and *worklist management* components causing the relative states of the two components to be synchronised.

4 Related work

There have been numerous papers advocating approaches to workflow and business process modelling based on Petri Nets (cf. [1],[10],[5],[18]), 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 [2],

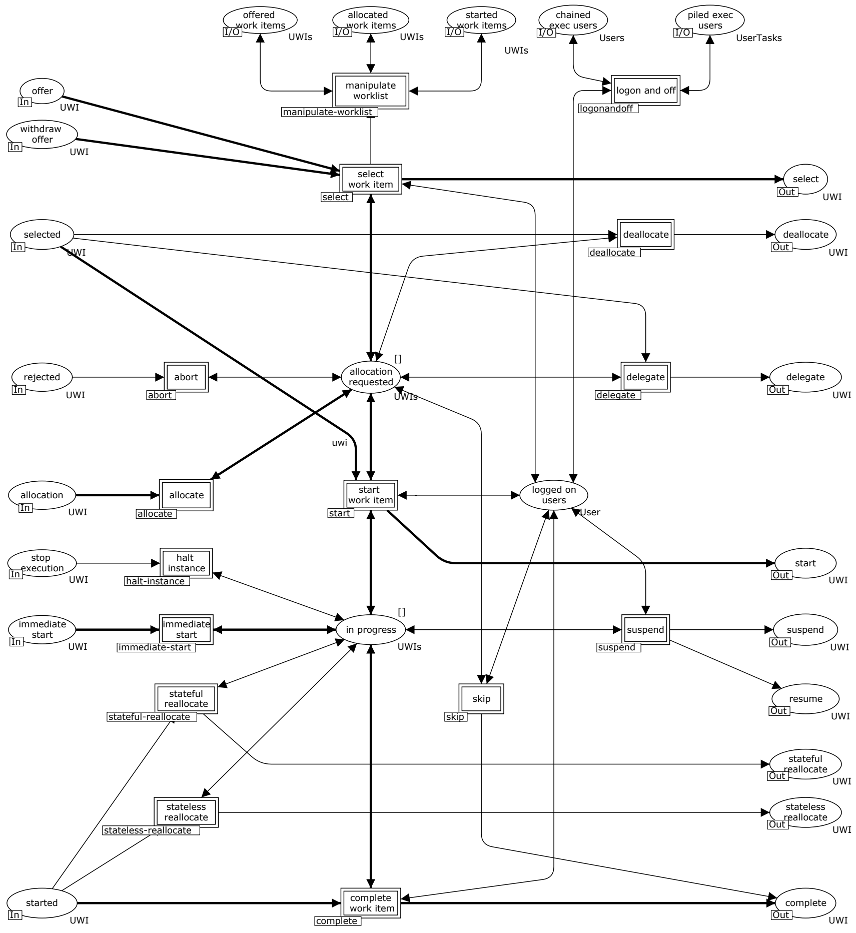

Fig. 7. *Worklist handler* component

UML 2.0 Activity Diagrams [24] and BPMN [8], 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 [19] 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 [7] and the direct animation of design-time models for requirements validation [17]. The latter of these approaches which uses a strategy based on Coloured Petri Nets [13] and CPN Tools [14] 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 [11], sequence diagrams [20] and task descriptions [15].

There has been a significant body of work that describes software architectures for workflow management systems. Significant examples of such systems include MOBILE [12], WIDE [6], CrossFlow [16] and WAMO [9] amongst many others however none of these systems offer a fully formalised description both of their language elements and the overall operation of the workflow system.

5 Experiences and conclusion

The *new*YAWL system model[3] *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 *new*YAWL. 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 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. *new*YAWL 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 system 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 2.1Ghz dual-core, 2Gb RAM), it takes almost 4 minutes just to load the model. Moreover the potentially infinite range of business process models that the *new*YAWL system can encode, rules out the use

[3] This model is available at `www.yawl-system.com/newYAWL`.

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 new*YAWL system model delivered some salient insights into areas of* new*YAWL that needed further consideration during the design activity.* These included:

- the introduction of a deterministic mechanism for recording status changes in the work item execution lifecycle in order to ensure that the views of these details maintained by the *worklist management* and *worklist handler* components are consistent;
- the establishment of a coherence protocol to ensure that reallocation of work items to alternate resources either by resources themselves or the workflow administrator are handled in a consistent manner in order to ensure that potential race conditions arising during reallocation do not result in the workflow engine, workflow administrator or the initiating resource (i.e. *worklist handler*) having irreconcilable views of the current state of work item allocations;
- the introduction of a consistent approach for handling the evaluation of any functions associated with a *new*YAWL 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 introduction of a locking strategy for data elements to prevent inadvertant side-effects of concurrent data usage; and
- 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.

The *new*YAWL system model provides a complete description of an operational environment for the *new*YAWL language. It is sufficiently detailed to be directly useful for system design and development activities. It will serve as the design blueprint for upcoming versions of the open-source YAWL System. The resource management component of the *new*YAWL language is currently being incorporated in the YAWL System.

References

1. 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.

2. W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.
3. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.
4. 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.
5. 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.
6. S. Ceri, P.W.P.J. Grefen, and G. Sanchez. Wide: A distributed architecture for workflow management. In *Proceedings of the Seventh International Workshop on Research Issues in Data Engineering (RIDE'97)*, Birmingham, England, 1997. IEEE Computer Society Press.
7. 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.
8. 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/.
9. J. Eder and W. Liebhart. The workflow activity model (WAMO). In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS-95)*, pages 87–98, Vienna, Austria, 1995. University of Toronto Press.
10. 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.
11. 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.
12. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation.* Thomson Computer Press, London, UK, 1996.
13. 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.
14. 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.
15. 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.
16. H. Ludwig and Y. Hoffner. Contract-based cross-organisational workflows - the crossflow project. In P. Grefen, C. Bussler, H. Ludwig, and M.C. Shan, editors, *Proceedings of the WACC Workshop on Cross-Organisational Workflow Management and Co-Ordination*, San Francisco, 1999.

17. 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.
18. 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.
19. 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.
20. 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.
21. N. Russell, A.H.M. ter Hofstede, and W.M.P. van der Aalst. *new*YAWL: Specifying a workflow reference language using Coloured Petri Nets. In *Proceedings of the Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, number DAIMI PB-584, pages 107–126. Department of Computer Science, University of Aarhus, Denmark, 2007.
22. 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.
23. 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.
24. 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 fur Informatik.