Int J Softw Tools Technol Transfer (2007) 9:327–352
DOI 10.1007/s10009-007-0036-z

SPECIAL SECTION CPN 04/05

Modelling work distribution mechanisms using Colored Petri Nets

Maja Pesic · Wil M. P. van der Aalst

Published online: 14 March 2007

Abstract Workflow management systems support business processes and are driven by their models. These models cover different perspectives including the control-flow, resource, and data perspectives. This paper focuses on the *resource perspective*, i.e., the way the system distributes work based on the structure of the organization and capabilities/qualifications of people. Contemporary workflow management systems offer a wide variety of mechanisms to support the resource perspective. Because the resource perspective is essential for the applicability of such systems, it is important to better understand the mechanisms and their interactions. Our goal is not to evaluate and compare *what* different systems do, but to understand *how* they do it. We use *Colored Petri Nets* (CPNs) to model work distribution mechanisms. First, we provide a basic model that can be seen as a reference model of existing workflow management systems. This model is then extended for three specific systems (Staffware, FileNet, and FLOWer). Moreover, we show how more advanced work distribution mechanisms, referred to as *resource patterns*, can be modelled and analyzed.

M. Pesic (✉) · W. M. P. van der Aalst
Department of Technology Management,
Eindhoven University of Technology,
P.O.Box 513, NL-5600 MB,
Eindhoven, The Netherlands
e-mail: m.pesic@tm.tue.nl

W. M. P. van der Aalst
e-mail: w.m.p.v.d.aalst@tm.tue.nl

1 Introduction

Workflow management systems are process-aware information systems [5,19], which are used in companies as a means for the computerized structuring and driving of complex business processes. Workflow management systems implement business process models, and use them for driving the flow of work by allocating the right employees to the right tasks at the right times. The system *manages the work of employees*. It will determine which tasks an employee has to execute and when, which documents will be used, which information will be available during work, etc. Typically, a workflow management system uses several mechanisms to distribute work. Nevertheless, we believe that existing systems are too limited in this respect. The goal of this paper is not to propose advanced work distribution mechanisms. Instead, we focus on the analysis of functionality in existing systems. The goal is not to evaluate these systems, but to understand *how* they offer specific functionality. A *deeper understanding* of particular aspects of work distribution is essential for developing a new breed of more user-centric systems.

The work reported in this paper can be seen as an extension of the *workflow patterns initiative*[1] [6]. Within the context of this initiative 43 resource patterns [48,46] have been defined. Using a patterns approach, work distribution is evaluated from the perspective of the end-user as a dynamic property of workflow management systems. The work reported in this paper adds to a better understanding of these mechanisms by providing explicit process models for these patterns, i.e., the

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

descriptive models are augmented with executable models. Most work reported in literature (cf. Sect. 4) uses static models to describe work distribution. Consider for example the meta modelling approaches presented in [8,38–40,45]. These approaches use static models (e.g., UML class diagrams) to discuss work distribution concepts. This paper takes a truly dynamic model—a *Colored Petri Net* model—as a starting point, thus clearly differentiating our contribution from existing work reported in literature.

Colored Petri Nets (CPNs) [29,32] are a natural extension of the classical Petri net [43]. There are several reasons for selecting CPNs as the language for modelling work distribution in the context of workflow management. First of all, CPNs have formal semantics and allow for different types of analysis, e.g., state-space analysis and invariants [30]. Second, CPNs are executable and allow for rapid prototyping, gaming, and simulation. Third, CPNs are graphical and their notation is similar to existing workflow languages. Finally, the CPN language is supported by CPN Tools[2] — a graphical environment to model, enact and analyze CPNs.

In this paper, we provide a basic CPN model that can be seen as a reference model of existing workflow management systems. The model will incorporate concepts of a task, case, user, work item, role and group. This model should be seen as a *starting point* towards a more *comprehensive reference model for work distribution*. The basic CPN model is extended and specialized for three specific systems: Staffware [51], FileNet [22], and FLOWer [41]. These three models are used to investigate differences between and similarities among different work distribution mechanisms in order to gain a deeper understanding of these mechanisms. In addition, advanced resource patterns that are not supported by these three systems are modelled by extending the basic CPN model.

The remainder of this paper is organized as follows. Section 2 presents the basic CPN model which should be considered as a reference model of existing workflow management systems. Section 3 extends this model in two directions: (1) Section 3.1 specializes the model for three different systems (i.e., Staffware, FileNet, and FLOWer), and (2) Sect. 3.2 extends the basic model for selected resource patterns. An overview of related work is given in Sect. 4. Section 5 discusses our findings and, finally, Sect. 6 concludes the paper.

[2] CPN Tools can be downloaded from http://www.wiki.daimi.au.dk/cpntools/.

2 Basic model

Different workflow management systems tend to use different work distribution concepts and completely different terminologies. This makes it difficult to compare these systems. Therefore, we will not start by developing CPN models for different systems and see how these can be unified, but, instead, start with modelling a reference model of existing systems. This model can assist in comparing systems and unifying concepts and terminology. We will use the term *Basic Model* to refer to this reference model and represent it in terms of a CPN model.

In the introduction we already motivated the use of CPNs as a modelling language [29,32]. A CPN consists of *places* and *transitions* connected by *arcs*. The network structure is static but places can hold *tokens* thus representing the state of the model. The number of tokens per place can vary over time. Moreover, unlike the classical Petri net, tokens can have both a value and a time-stamp. The time-stamps indicate the availability of tokens and can be used to model delays, processing times, timeouts, etc. The value of a token indicates the properties of the object represented by this token. Places (represented by ovals) are typed, i.e., the tokens in a place have values of a particular type (or color in CPN jargon). These types are a subset of the data types in Standard ML such as the primitive types integer and string and compositional types such as tuple, list and record. Each place can hold tokens with values of a certain type. Transitions (represented by rectangles) may consume and produce tokens. Since tokens have values, *arc inscriptions* are needed to specify the input-output relations. Besides the extension with token colors and time-stamps, CPN models allow for hierarchy. Complex models may be decomposed into sub-pages, also referred to as sub-processes or modules, to obtain a layered hierarchical description. A more detailed discussion of the CPN concepts is beyond the scope of this paper. In the remainder, we assume that the reader is familiar with the CPN language and refer to [29,32] for more details.

We developed the Basic Model as a work distribution model of an example of a typical workflow management systems presented in Chap. 2 and 3 in the workflow management book [5]. The Basic Model represents a workflow management system where the business *process* is defined as a set of *tasks*. Before the process can be initiated and executed, it has to be instantiated. One (executable) instance of a process is referred to as a *case*. Each case traverses the process. If a task is enabled for a specific case, a *work item*, i.e., a concrete piece of work, is created. There is a set of *users* that can execute work items. The users are embedded in the

Table 1 Basic workflow concepts

```
color Task = string;
color Case = int;
color WI = product Case * Task;
color User = string;
color Role = string;
color Group = string;
```

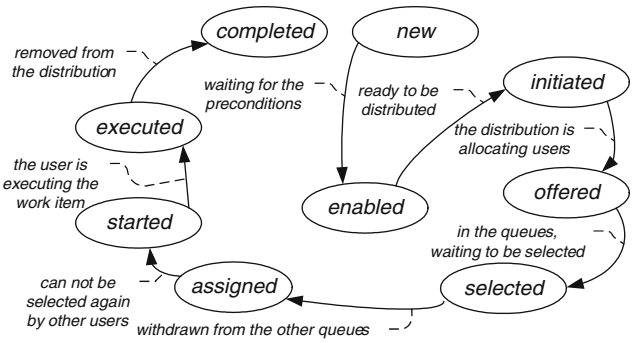

Fig. 1 Basic model—work item life cycle

organizational structure on the basis of their *roles*, and the *groups* they belong to. Group is an organizational unit (e.g., sales, purchasing, production, etc.), while role represents a capability of the user (e.g., manager, software developer, accountant, etc.). These concepts are mapped onto CPN types as shown in Table 1. As indicated, CPN uses Standard ML types (e.g., *string* and *int*) and type constructors such as *product* to create pairs and other complex constructs (e.g., (1,"*taskA*") represents a value of type *WI*).

During the work distribution work items change state. The change of state depends on the previous state and determines the next actions of users and the distribution mechanism. A model of a life cycle of a work item shows how a work item changes states during the work distribution. For more detailed models about life cycle models we refer the reader to literature, e.g., [5,17,19,28,35,40]. We develop and use the life cycle models as an aid to describe work distribution mechanisms. The Basic Model uses a simple model of the life cycle of work items and it covers only the general, rather simplified, behavior of workflow management systems (e.g., errors and aborts are not considered). Figure 1 shows the life cycle of a work item of the Basic Model. After the *new* work item has arrived, it is automatically also *enabled* and then taken into distribution (i.e., state *initiated*). Next, the work item is *offered* to the user(s). Once a user *selects* the work item, it is *assigned* to him/her, and (s)he can *start* executing it. After the *execution*, the work item is considered to be *completed*, and the user can begin working on the next work item.

To simulate (execute) the work distribution model, it is necessary to initiate the model by defining *input elements*. Table 2 shows the four elements that are required for the simulation of the Basic Model. For every input element, Table 2 shows the element name (i.e., "system users", "new work items", "task maps" and "user maps"). Besides the name, there are a short description of the element, the color in the CPN model that represents the element and a simple example of the initial element value. In this example, there are two work items available for the case "1": "write article" and "read article" (*new work items*). The authorization (*task maps*) of these two tasks is specified in such a way that the task "write article" is mapped to the user who has the role "student", and is in the group "Information Systems". The task "read article" is mapped to the user with the role "professor", from the group "Information Systems". The organizational structure (*user maps*) contains two *users*. First, there is "Mary" who has the role of "student" in the group "Information Systems". Second, user "Joe" has the role "professor" and he works in the groups "Information Systems" and "Mathematics".

Table 2 Input for the basic model

1.	**System users**	a set of available users;
	CPN color:	color Users = list User;
	example:	iUser = 1`"Mary"++1`"Joe";
2.	**New work items**	work items that have arrived and are ready to be distributed to users;
	CPN color:	color WI = product Case * Task;
	example:	iWI = 1`(1,"write article")++1`(1,"read article");
3.	**Task maps**	for every task authorization is defined with a role and a group;
	CPN color:	color TMap = product Task * Role * Group;
	example:	iTMaps = [("write article", "student", "Information Systems"), ("read article", "professor", "Information Systems")];
4.	**User maps**	the organizational structure is used to map users to the authorization of tasks;
	CPN color:	color UMap = product User * Roles * Groups; (color Roles = list Role; color Groups = list Group;)
	example:	iUMaps = [("Mary", ["student"], ["Information Systems"]), ("Joe", ["professor"], ["Mathematics", "Information Systems"])];

As a model of an abstract workflow management system, we have developed the Basic Model on the basis of predefined assumptions: (1) we abstract from the process perspective (i.e., splits, joins, creation of work items), (2) we only consider the "normal" behavior (i.e., work items are completed successfully; errors and aborts are not included), and (3) we abstract from the user interface.

The Basic Model is organized into two sub-systems: the Work Distribution and the Work Lists module. The CPN language allows for the decomposition of complex nets into sub-pages, which are also referred to as sub-systems, sub-processes or modules. By using such modules we obtain a layered hierarchical description. Figure 2 shows the modular structure of the Basic Model. The two sub-modules communicate by exchanging messages via six places. These messages contain information about a user and a work item. Every message place is of the type (i.e., the CPN color set) "user work item" (*color UWI = product User * WI*), which is a combination of a user and a work item. Table 3 shows the description of the semantics of different messages that can be exchanged in the model.

Work distribution The work distribution module manages the distribution of work items by managing the process of work execution and making sure that work items are executed correctly. It allocates (identifies) users to whom the *new work items* should be offered, based on authorization (*TMap*) and organization (*UMap*) data. Three (out of four) input elements are placed in this module: *new work items*, *user maps* and *task maps*.

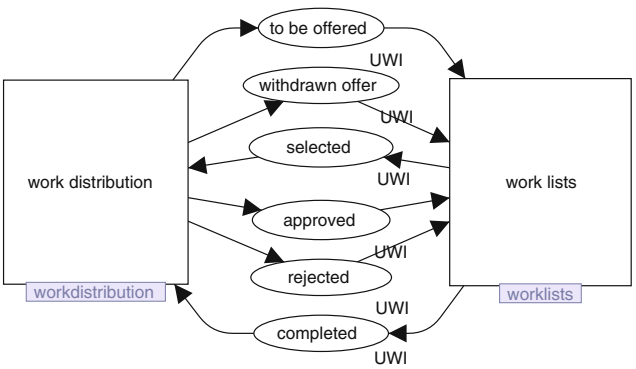

Fig. 2 Basic model—main

Table 3 Messages between modules

Place	Message
to be offered	A work item is offered to the user.
withdrawn offer	Withdraw the offered work item from the user.
selected	The user requests to select the work item.
approved	Allow the user to select the work item.
rejected	Do not allow the user to select the work item.
completed	The user has completed executing the work item

Figure 3a shows the Work Distribution module. The new work items are determined as input values (initial marking) in the place *new work items*. The first to fire is the transition *offers*, which retrieves the task maps and user maps (as two input elements) from the places *task map* and *user map*. These three input parameters are passed to the function *offer* on the outgoing arc which produces user work items in the place *to be offered*. A user work item in the place *to be offered* sends the message to the Work Lists module to offer the work item to the user. This transition removes the work item token from the place *new work items* and adds it to the list of work items in the place *offered work items*, which stores all work items that are offered but not selected yet. This procedure is followed with every work item token from the place *new work items*.

The core and the logic of the allocation is captured in the allocation function *offer*, in the inscription of the outgoing arc from the transition *offers* to the place *to be offered*. This function contains allocation rules (allocation algorithm) of the specific distribution mechanism. Based on these rules and the three input arguments it determines which work items should be offered to which users (*color UWI = User * WI*). The function *offer* uses three input parameters: (1) user maps (*var umaps: UMaps*), (2) task maps (*var tmaps: TMaps*), and (3) a new work item (*var wi: WI*). The allocation function *offer* of the Basic Model searches for the users to offer the work item in several steps: (1) decompose the input parameter for the work item (*wi*) into two elements - a case (*c*) and a task (*t*); (2) in the input parameter task maps (*tmaps*) find a task map for the referring task (*t*); (3) from the found task map get the role (*r*) and the group (*g*); (4) in the input parameter user maps (*umaps*) find all users that have both the referring role (*r*) and are in the referring group (*g*); and (5) for every user that was found, create an offer—an user work item token consisting of the referred user value and the referred work item value. Thus, the allocation function in the Basic Model offers the new work item to all the users that have the authorized role and group to execute the task.

After the work item is allocated, the offers to users are sent to the Work Lists module via place *to be offered*, and stored in the place *offered work items*. Next, the Work Distribution module waits for the message from the Work Lists module that a user requests to select (and further execute) a work item. This message arrives as a user work item in the place *selected*. The message

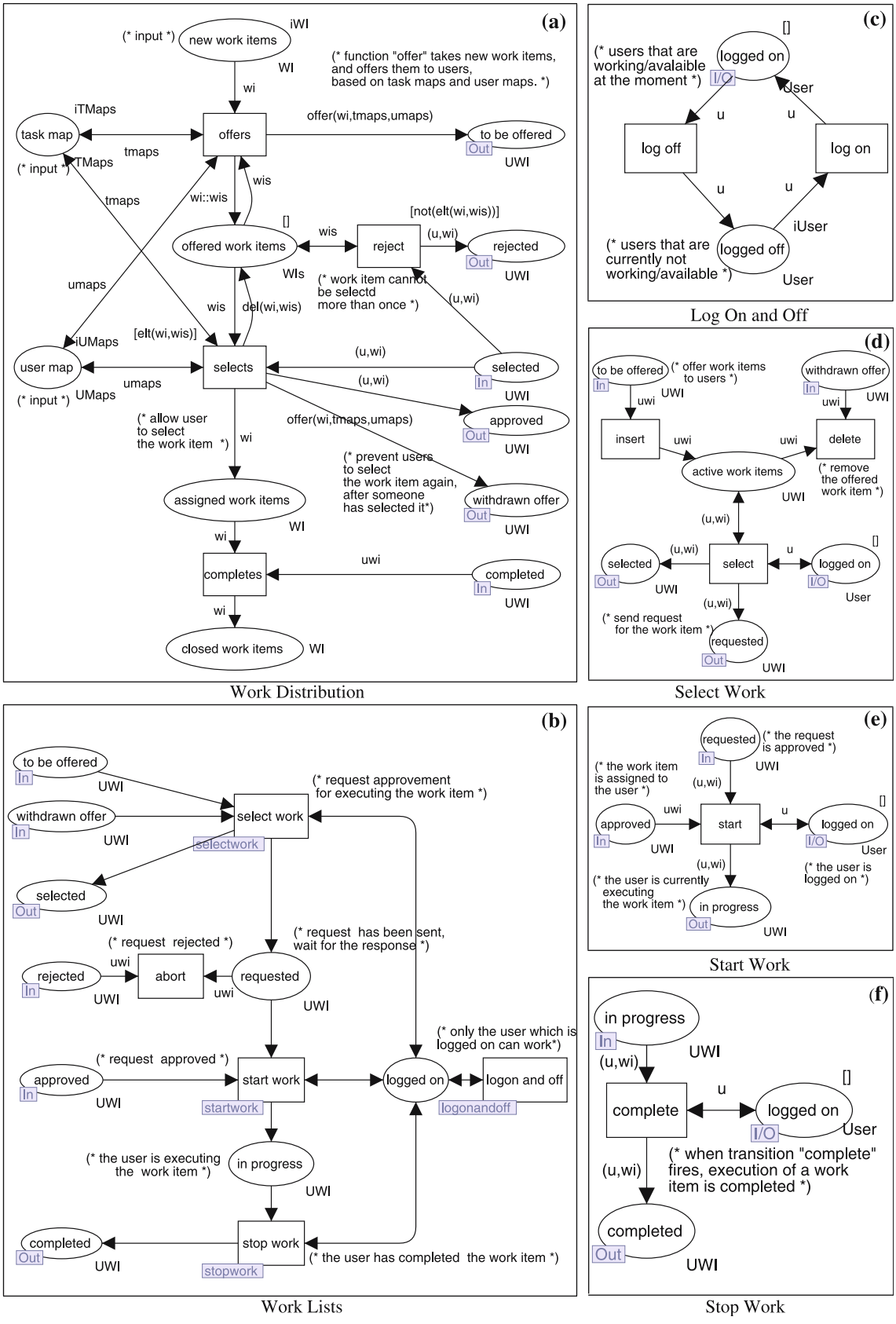

Fig. 3 Basic model

(token) contains the information about the work item and the user that requests to select it.

The work distribution module makes decision about the selection request based on the principle that only one user at one moment can select (and execute) a work item. Thus, the place *selected* is connected with incoming arcs to the transitions *selects* and *reject* and when a token arrives at this place only one of these two transition will fire. If the referring work item is in the place *offered work items* this means that it has not been selected yet and the selection request should be approved, i.e., the transition *selects* should fire. This is achieved with the guard on the transition *selects*, which makes sure that this transition can fire only if the work item is in the list of work items in the place *offered work items*. This guard contains the function *elt* which checks if a list contains an element. The transition *selects* removes the work item from the list in the place *offered work items* via a simple function *del*, which deletes an element from the list. It also removes the token from the place *selected* to remove the request and puts a user work item token in the place *assigned work items* and in the place *approved*. If a user work item is stored in the place *assigned work items*, this means that the referring user has selected the referring work item. A user work item token in the place *approved* sends a message to the Work Lists module that the referring user can select the referring work item. Since the Work Distribution module can offer a new work item to multiple users and follows the rule that only one user can select and execute a work item at one time, when one user selects a work item, all offers should be withdrawn. This is why the transition *selects* uses again the allocation function *offer* (with the same parameters as the transition *offers*) to crate a set of work item tokens that were previously offered (by the transition *offers*) in the place *withdrawn offer*. A user work item token in the place *withdrawn offer* sends a message to the Work Lists module to withdraw the offer of the referred work item from the referred user.

If the work item has already been selected by a user, it is not in the place *offered work items* and the request should be rejected, i.e., the transition *reject* should fire. For this purpose we check if the work item is not the list in the place *offered work items* (with the negation of the function *elt*) in the guard of the transition *reject*. This transition removes the user work item token from the place *selected* (in order to remove request) and puts it in the place *rejected*. A user work item token in the place *rejected* sends a message to the Work Lists module to reject a request of the referred user to select the referred work item.

A message from the Work Lists module that a user has completed a work item arrives with a user work item token in the place *completed*. The transition *completes* matches the user work item tokens in the places *completed* and *assigned work items*, removes them from those two places, and produces the referring user work item token in the place *closed work items*. This user work item is considered to be completed by the user, and it is archived as closed.

Work lists Figure 3b shows the work lists module. This module receives messages from the Work Distribution module about which work items are to be offered to which users. The Work Lists module further manages events associated with the activities of users. It is decomposed into three units, which correspond to three basic actions users can make: *log on and off* (cf. Fig. 3c) in the system, *select work* (cf. Fig. 3d), *start work* (cf. Fig. 3e), and *stop work* (cf. Fig. 3f).

In the sub-module Log On and Off (cf. Fig. 3c) every user can freely choose when to log "on" or "off" in the system. When the transition *log on* fires, a token representing a user is moved from the place *logged off* and produced in the place *logged on*. When the transition *log off* fires, a token representing one user is moved from the place *logged on* to the place *logged off*. Thus the users of the system can either be logged on or off in the system. These two states of a user are represented by an appropriate token either in the place *logged on* or *logged off*, respectively. In order to perform any action in the Work List module, it is necessary that the user is *logged on* in the system.

Once the work item has been *offered* to users and a token is in the place *to be offered*, the Select Work sub-module (cf. Fig. 3d) automatically fires the transition *insert* and moves the user work item token to the place *active work items*. From this place, the users who are logged on (represented by appropriate tokens in the place *logged on*), can choose work items that are offered to them by firing the transition *select*. This transition consumes (removes) a token from the place *active work items*. The token is of the type *user work item* and its user value matches information about the user in the place *logged on*. This assures that only users that are currently logged on can select work. When the transition *select* matches the *logged on* user to the user work item token in the place *active work items* it creates tokens for this user work item in the places *selected* and *requested*. By creating a token in the place *selected*, a message about the intention (request) of the user to select this work item is sent to the Work Distribution module. When a token is placed in the place *requested*, the Work Lists module stores the information about the request that is sent and waits for the reply from the Work Distribution module. In the case that the message

to withdraw offer arrives at the Work Lists module and Select Work sub-module, it is represented by a token in the place *withdrawn offer*. The Select Work sub-module then automatically fires the transition *delete*, which removes a token from the places *active work items* and *withdrawn offer* and removes the offered work item from the Work Lists module.

The Work Lists module (cf. Fig. 3b) proceeds with the user work item in the place *requested* following one of the two alternative scenarios. Which scenario will be executed depends on the answer (i.e. on the selection request) that arrives from the Work Distribution module. If the message (a user work item token) arrives at the place *rejected*, the transition *abort* automatically fires and removes the token from the places *rejected* and *requested*. This means that the request from the user to select a work item is rejected and (s)he can not start working. If the message arrives at the place *approved*, the user can select the work item and further flow is directed to the Start Work sub-module.

In the Start Work sub-module (cf. Fig. 3e) the transition *start* fires if it matches the tokens for a user work item in places *requested* and *approved* and the referring user token is in the place *logged on*. When this match is achieved, the tokens are removed from the places *requested* and *approved* and a user work item token is placed in the place *in progress*. While the user is executing the work item the token remains in the place *in progress*.

The Stop Work sub-module (cf. Fig. 3f) has one transition *complete*, which fires when the user (represented by a token in the place logged on) completes a work item that is in progress. The transition *complete* removes the token from the place *in progress* and puts it to the place *completed*. The place *completed* is the last place for a user work item in the Work Lists module, and when a token gets in this place, a message is sent to the Work Distribution module that a user has completed a work item.

2.1 Evaluation

A state space analysis of the Basic Model is used for the analysis of the correctness of the model. The results of the state space analysis can be interpreted to check the extend to which the model complies with the properties of work distribution in workflow management systems. The state space analysis of the Basic Model was performed in the CPN Tools [30]. The state space analysis of the original model could not be successfully completed because the original model is not bounded. The place *requested* in the Work Lists module is not bounded (cf. Fig. 3b), i.e., the user can request a work item indefinitely often until the reply about the selection arrives from the Work Distribution module. The Logging on and off sub-module greatly increased the necessary time to construct the state space. Therefore, we changed the model at two points. First, when a user requests a work item, (s)he has to wait for the Work Distribution module to reply (and remove the referring token from the place *request*) before s(he) can request the same work item again. This means that there cannot be more than one work item token for the same user in the place *request* at one moment—this place is bounded. Second, to improve the speed of the state space construction, we have removed the Log On and Off sub-module and we assume that all the users are logged on to the system at any moment. These changes made it feasible to analyse the state space constructed by the CPN tools and did not jeopardize the principles of the work distribution in the Basic Model.

Boundedness properties The state space analysis provides the upper and lower integer bound of tokens in every place. The bounds depend on the given initial marking. For the state space analysis the initial marking consisted of two tokens in the place *new work items*, one token in the places task map and user map (each) in the Work Distribution module and two tokens in the place *logged on* in the Work Lists module. The boundedness results are shown in Table 4. As expected, all places have an upper bound (the upper bound is at most three, because we changed the model to be bounded. Most of the places have zero as the lower bound, except for the places *offered work items*, *task map*, and *user map*, *requested* and *logged on*. The places *offered work*

Table 4 Integer bounds of a simplified basic model

Module	Place	Bound	
		Up	Low
Main	Approved	2	0
Main	Completed	2	0
Main	Rejected	3	0
Main	Selected	3	0
Main	To be offered	3	0
Main	Withdrawn offer	3	0
Select work	Active work items	3	0
Work distribution	Assigned work items	2	0
Work distribution	Closed work items	2	0
Work distribution	New work items	2	0
Work distribution	Offered work items	1	1
Work distribution	Task map	1	1
Work distribution	User map	1	1
Work lists	In progress	2	0
Work lists	Logged on	2	2
Work lists	Requested	1	1

items, *task map*, and *user map*, and *requested* each use a list to store tokens, and thus always contain one token (the list). Therefore, these places have both lower and upper bound of one token. Because we keep all user tokens at all times in the place *logged on*, this place also has equal upper and lower bound. This is why the two user tokens from the initial marking always remain in this place. Places that refer to an work item that is in some stage of the execution should all have the upper bound the same like the number of tokens in the place *new work items* in the initial marking, i.e., two tokens. This is because every work item will be executed exactly once. The Basic Model satisfies this property, since places *new work items, approved*, *assigned work items*, *in progress*, *completed*, and *closed work items* all have the upper bound of two tokens. Places *rejected*, *selected*, *to be offered*, *withdrawn offer* and *active work* items have the upper bound of three tokens, because the two initial tokens in the place *new work items* result in three offered user work item tokens in the place *to offer*.

Home properties The Basic Model assumes that all the new work items will be executed by the users and that, after a finite execution sequence, the tokens from the place *new work items* will be produced in the place *closed work items*, without any "work item" related tokens left in the net. Thus, none of the markings would always be possible to reach, i.e., there are no home markings in the Basic Model.

Liveness properties The state space analysis reported a number of dead markings. i.e., a number of states of the net in which no transition is enabled. This is a desired property of the Basic Model, because at every marking where the place *new work items* reaches the lower bound and the place *closed work items* reaches the upper bound, no other transition in the net should be enabled, because all the work items were completed and closed.

3 Work distribution models

The Basic Model presented in previous section (Sect. 2) is used as a reference for different extensions and specializations of work distribution. In this section, we first extend and specialize the Basic Model to accommodate the capabilities of Staffware, FileNet and FLOWer (Sect. 3.1). In Sect. 3.2 we select four of the more advanced resource patterns reported in [46,48]. These four patterns are not supported by Staffware, FileNet and FLOWer, but we will show that it is easy to extend the Basic Model to adequately address the patterns.

3.1 Workflow management systems

We have modelled the work distribution mechanisms of three commercial workflow management systems: Staffware, FileNet and FLOWer. FileNet and Staffware are examples of two widely used traditional workflow management systems. FLOWer is based on the case-handling paradigm, which can be characterized as "the more flexible approach" [3,9]. Each of the models we have developed will be described in the remainder of this section.

3.1.1 Staffware

The Basic Model is extended to represent the work distribution of Staffware. The way of modelling the *organizational structure* and *resource allocation algorithm* are changed, while the concept of *work queues* and the possibility of the user to *forward and suspend* a work item are added to the model. In this section we first describe the organizational structure of Staffware. Second, we describe the work queues and the two level distribution that accompanies them. Third, we explain the resource allocation of Staffware and its allocation function. Finally, we show which features have to be added to the Basic Model to implement the suspension and forwarding of work.

Organizational structure Simple organizational structure can be created in Staffware using the notions of *groups* and *roles*. The notion of group is defined as in the Basic Model, i.e., one group can contain several users, and one user can be a member of several groups. However, specific in Staffware is that a role can be defined for only one user. This feature does not require any changes in the model structure or color sets. However, it changes the way the initial value for the *user maps* should be defined – one role should be assigned to only one user.

Work queues Groups are used in Staffware to model a set of users that share common rights. The work item can be allocated to the whole group, instead of listing the names of users that can execute it. Staffware introduces a *work queue* for every group. The work queue is accessible to all members of the group. Single users can be considered to be groups that contain only one member. Thus, one work queue is also created for every user and this personal queue is only accessible by a single user. From the perspective of the user, (s)he has

access to the personal work queue and to work queues of all the groups (s)he is a member of. While the Basic Model (Sect. 2) offers the work item directly to *users*, Staffware offers items in two levels. First, a work item is offered to *work queues* (*color WQ = string*) in the Work Distribution module (cf. Fig. 4). We refer to this kind of work items as to *queue work item* (*color QWI = product WI * WQ*). Second, every queue work item is offered to the members of a group (work queue) in the Offering sub-module (cf. Fig. 5). Only one member will execute the queue work item once. We refer to a queue work item that is offered to a member (of a work queue) as to *user work item* (*color UWI = product User * QWI*).

Figure 4 shows the first level of distribution in the Work Distribution module of Staffware. The transition *offers to work queues* removes a work item token from the place *new work items* and creates offers to work queues by producing queue work item tokens in place *to offer to work queues*. To do this, it retrieves task maps, user maps and field maps as input elements. It also produces a work item token in the place *offered work items*. The queue work item tokens in the place *to offer to work queues* are produced by the allocation function *offer_qwi* in the arc inscription between the transition *offers to work queues* and the place *to offer to work queues*. This function takes a work item, task maps, user maps and field maps[3] as parameters. The effects of this function are explained in the paragraph Resource Allocation of this section (Sect. 3.1.1). The transition *offers to work queues* produces a queue work item token in the place *offered work items* to store the information about which work items are expected to be completed by work queues. A token in the place *to offer to work queues* sends a message to the Offering sub-module that the queue work item should be further distributed to the work queue members. After the completion of a queue work item, the Offering sub-module sends a message by creating a queue work item in the place *completed queue work items*. The transition *completes work item* considers a work item to be completed when all queue work items that originate from that work item are completed. The transition retrieves a work item from the place *offered work items* and waits until all queue work items that originate from (that were offered to work queues based on) the referring work item. For this reason, the allocation function *offer_qwi* is called on the arc inscription between the place *completes work item* and the transition *completes work item* with the

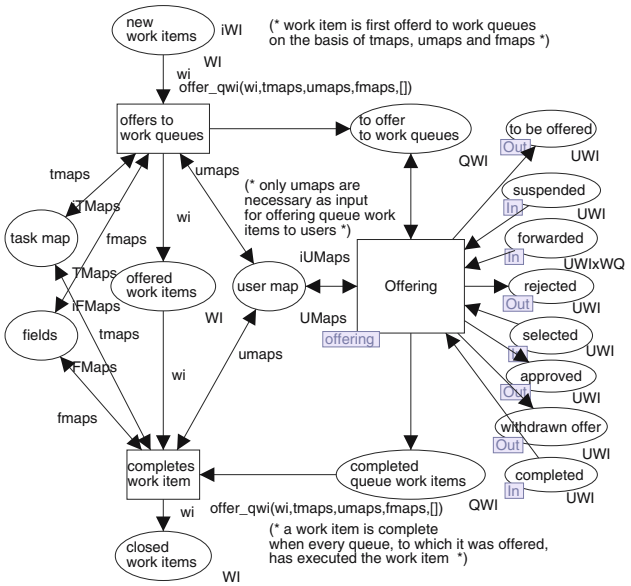

Fig. 4 Staffware—work distribution

same parameters like in the arc inscription between the transition *offers to work queues* and the place *to offer to work queues*. Distribution to work queues in Staffware follows a similar logic like the distribution in the Basic Model, but also introduces some changes. A difference between these two distribution models is that, instead of distributing work directly to the Work Lists module (users) like in the Basic Model, the Staffware Work Distribution module hands-off the distribution to users to is sub-module Offering. While a work item is the object of distribution in the Basic Model, the Staffware Work Distribution module distributes queue work items.

Figure 5 shows the second level of distribution in the Offering sub-module of Staffware. The first transition to fire here is the transition *offers to work queues*, when the message about the new queue work item is received from the Work Distribution module. This message is received when a new queue work item token arrives at the place *to offer to work queues*. This transition removes the queue work item from the place *to offer to work queues* and produces it in the place *offered work queues*, retrieves user maps and creates new user work items in the place *to be offered*. The offers for users are created by the allocation function *offer_uwi*, which takes a queue work item that is to be offered and the user maps as parameters. This function searches in user maps for all members of the work queue and creates a user work items for each member that was found. The Offering sub-module follows the logic of the the Basic Model Work Distribution module. For a detailed description of this kind of distribution we refer the reader to the Sect. 2. However, instead of starting with work items like the

[3] The fifth parameter is an empty list and is used as aid to perform calculations in the function. This parameter should always be left empty and does not influence the function results.

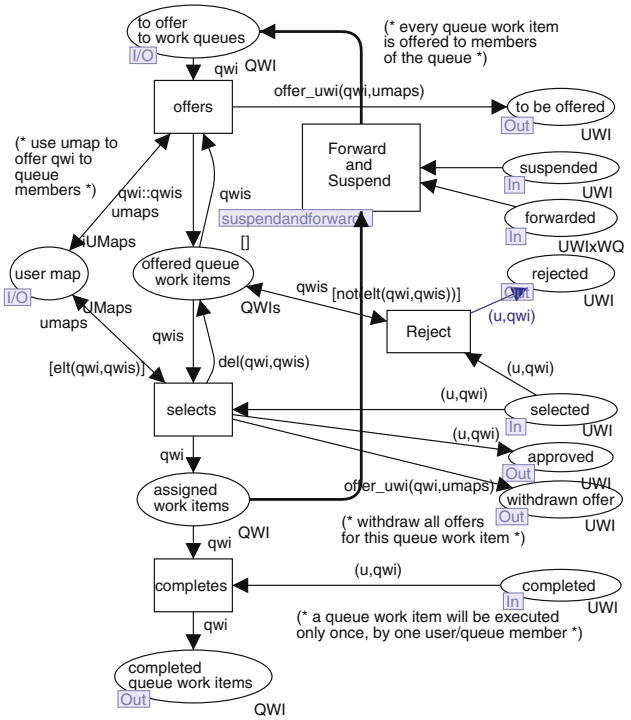

Fig. 5 Staffware—offering

Basic Model, the Offering sub-module starts with available queue work items. An addition to the Staffware model was the possibility to suspend and forward work. These mechanisms were added in the Suspend and Forward sub-module, which will be explained later in this section.

Resource allocation The resource allocation of Staffware is captured in the two level distribution mechanism with two allocation functions: (1) function *offer_qwi* (cf. Fig. 4) takes a new work item, task maps, user maps and field maps as parameters and allocates work queues that are authorized to execute the work item; (2) function *offer_uwi* (cf. Fig. 5) takes a queue work item and user maps as parameters and allocates all users that are members of the referring queue.

Just like the Basic Model, Staffware searches for possible users based on *roles* and *groups*. In addition to this, in Staffware users can be allocated by their *user names* and *data fields* in the process. Thus, task maps in the Staffware model assign a list of users, roles, groups and fields to each task (*TMap = product Task * Users * Roles * Groups * Fields*). Figure 6 shows how a task map is specified in Staffware. Based on task maps, function *offer_qwi* (cf. Fig. 4) allocates work queues that are authorized to execute the work item: (1) when a user name is provided in a task map, the work item is offered to personal work queue of the referring user;

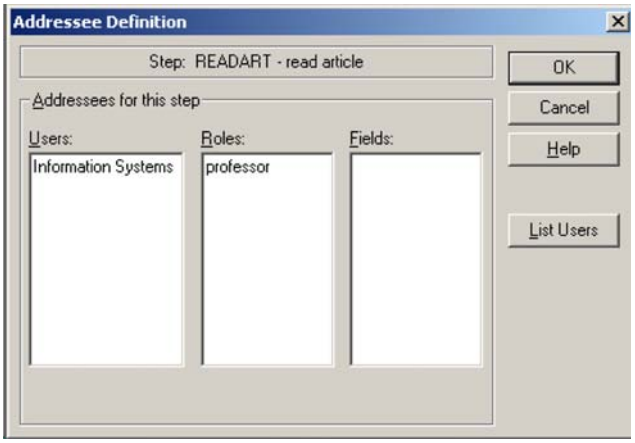

Fig. 6 Staffware—a task map

(2) for every role in the task map, this function offers the work item to the personal work queue of the user with that role (note that one role can be assigned to only one user); (3) a work item is offered to the work queue of every group that is stated in the task maps; and (4) for authorizations via fields, allocation is executed at the run-time. The allocation at run-time is referred to as a dynamic work allocation. Every field has a unique name (*color Field = string*), e.g., *"next user"*. During the execution of the process, every field is assigned a value, and this value changes (e.g., users can assign values to fields). Staffware assumes that the value of the assigned data field is a group name, a role name or a user name. If the field *"next user"* (which for example has the value of "Joe Smith" assigned) is specified in the task map of a task, then the actual value of the field is assigned to the task map entry at the moment when the task becomes enabled. Thus, "Joe Smith" will be used in the allocation. Figure 7 shows Staffware Process Client tool, where users can access their work queues and process the work items. In this case, there are two work queues: (1) the work queue for the group "Information Systems", and (2) the personal work queue of the user "Joe".

When all the properties of the Staffware work distribution are merged together, unexpected scenarios might

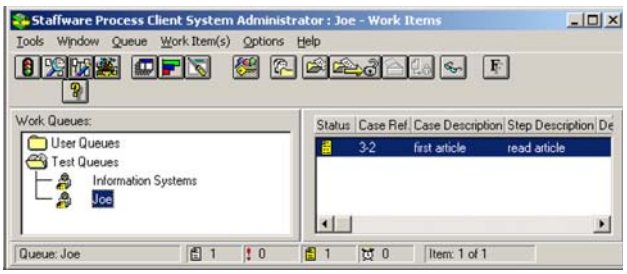

Fig. 7 Staffware—a work queue with a work item

happen. If we look at the example from Table 2, we can see that the task "read article" should be allocated to users which are from the group "Information Systems" *and* have the role "professor". The Basic Model allocates this task to users that are from the group "Information Systems" *and* have the role "professor", i.e., to the user "Joe". Unlike the Basic Model, Staffware allocates this task to: (1) the work queue of the group "Information Systems" (which members are "Mary" and "Joe"), and (2) the personal queue of the user who has the role "professor" (with one member "Joe"). A work item is completed in Staffware when all its queue work items are completed (cf. Fig. 4). Thus, the task "read article" will be execute two times: (1) once by a member of the of the group "Information Systems"—"Mary" or "Joe", and (2) once by the user who has the role "professor"—"Joe". As the result of Staffware work distribution, the work item "read article" has two possible scenarios of the execution. This task will be executed either once by "Mary" and once by "Joe", or two times by "Joe". Which one of these two scenarios will take place, depends only on which user is faster, i.e., on which users select the task before the others do.

Forward and suspend When the user selects a work item in the Basic Model, the work item is *assigned* to him/her, and (s)he can *start* the work item and *execute* it. Figure 8 shows that Staffware offers a more realistic and somewhat more complex model of the life cycle of a work item than the Basic Model. After the user *selects* the work item, it is *assigned* to him/her, and then (s)he can either *start* the work item or *forward* it to another user. Forwarding transfers the work item to the state *offered*, because it is automatically *offered* to the new user. If the user chooses to *start* the work item, (s)he can *execute* it or *suspend* it. When a work item is suspended, it is transferred back to the state *initiated*. After this, the system *offers* the work item again to all authorized users.

Forwarding and suspending of work items adds two messages that are exchanged between Work Distribution and Work Lists modules in Staffware model. Figures 4 and 5 show two new places—*forward* and *suspend*. These two new actions are triggered in the *Work List* module by the user.

Figure 9 shows that in the Staffware sub-module Start Work the user can choose to *select* or *forward* (to another work queue) the work item. To enable forwarding, we add the transition *forward* to the Start Work sub-module in Staffware model. The request to select a work item is represented with a user work item in the place *requested*. After this request, the Start Work sub-module waits until the Work Distribution module approves the request, by creating a user work item token in the place *approved*. When the request is approved the transitions *start work* and *forward* can fire depending on the user decision. Both transitions consume the two matching user work item tokens from the places *requested* and *approved*. The transition *start work* has the same effect as the Basic Model. The transition *forward* matches the user token in the place *logged on* with the referring user work item, retrieves a work queue token from the place *work queues* and produces a token in the place *forwarded*. The initial marking for the place work queues consists of all group names and all user names registered in the system. This is straightforward because Staffware creates group work queues for all groups and personal work queues for all users. The place *forwarded* is of the color set type that combines a user work item and a work queue to which the work item should be forwarded(*color UWIxWQ = product UWI*WQ*). The transition *forward* produces a token in the place *forwarded* with the arc inscription *((u,qwi),wq)*. This token is sends the message to the Work Distribution module that the referring user (*u*) wants to forward the referring queue work item (*qwi*) to the referring work queue (*wq*).

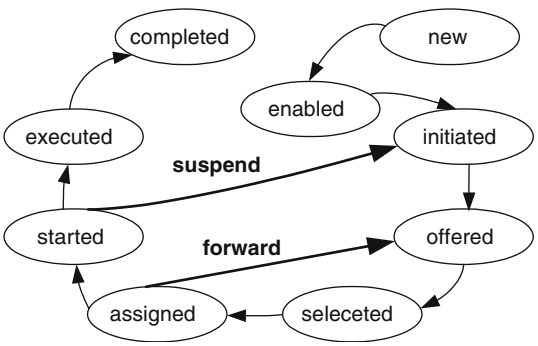

Fig. 8 Staffware—work item life cycle

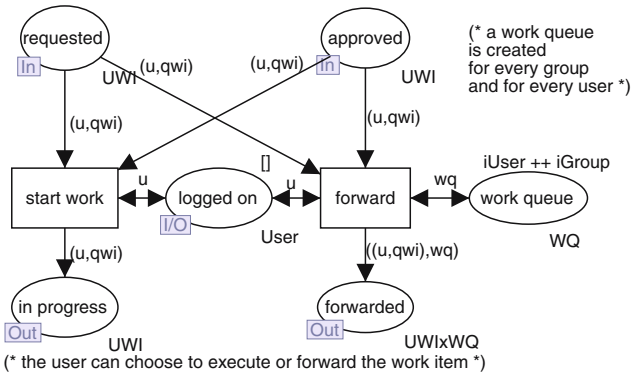

Fig. 9 Staffware—start work

Figure 10 shows that in the sub-module Stop Work the user can choose to *complete* or *suspend* the work item. The transition *suspend* is added to the sub-module. While a user is executing a queue work item, a referring user work item token is in the place *in progress*. At any time during the execution of a work item, one of the transitions *complete* and *suspend* can fire. While transition *complete* has the same effects as in the Basic Model, transition *suspend* is new in Staffware model. This transition matches the user token in the place *logged on* with the user work item in the place *in progress*. It consumes the user work item from the place *in progress* and produces the referring user work item token in the place *suspended*. A user work item token in the place *suspended* sends the message to the Work Distribution module that the referring user wishes to suspend the referring queue work item.

The Work Distribution module handles forwarding and suspending in a new sub-module: the Suspend and Forward sub-module (cf. Fig. 11). This Figure shows how: (1) in case of forwarding the work item is automatically *cancelled* for the current work queue and *offered* to the new work queue, and (2) in case of suspending the work item is *cancelled* for the current work queue and *re-offered* as a new work item. When a message that a user wishes to forward a queue work item to a work queue from the Work Lists module arrives, a token is produced in the place *forwarded*. The Forward and Suspend sub-module then automatically fires the transition *forward*. This transition consumes the token from the place *forwarded* and produces two different tokens in places *to re offer* and *to cancel*. The queue work item token that is forwarded is produced in the place *to cancel*. A new queue work item, which consists of a referring work item and a new work queue, is produced in the place *to re offer*. When the message that a user wishes to suspend a user work item a token is produced in the place *suspended*. The transition *suspend* fires automatically when the message arrives, consumes the user work item token from the place *suspended* and produces two identical referring queue work item tokens in the places *to cancel* and *to re offer*. The transitions *re offer* and *cancel* fire automatically when tokens are produced in places *to re offer* and *to cancel*, respectively. Transition *cancel* consumes matching queue work item tokens from the places *to cancel* and *selected work items*. In this way the queue work item is removed from the model. The transition *re offer* consumes a queue work item token from the place *to re offer* and produces one in the place *to offer to work queues*. In this way, the Offering sub-module can offer the queue work item to the members of the work queue again.

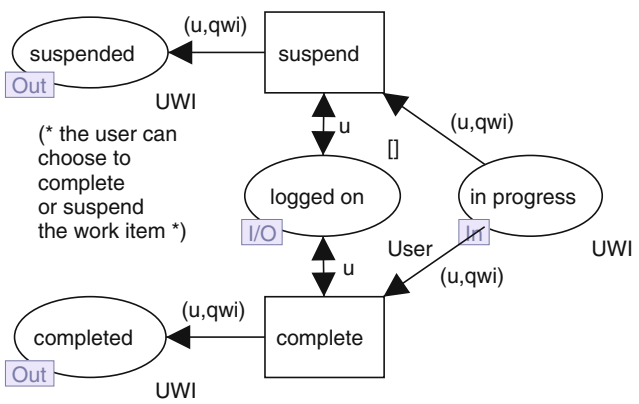

Fig. 10 Staffware—stop work

3.1.2 FileNet

Like Staffware, FileNet is a widely used traditional process-oriented workflow management system. In this section we will describe the FileNet CPN model that we develop using the Basic Model as a starting reference model.

Organization Unlike the Basic Model and Staffware, FileNet does not allow for modelling *roles* of users. The organizational structure in FileNet can be modelled via two types of groups:

1. Administrators of the FileNet system can define *work queues* (*color WQ = string*) and assign their members by selecting users of the FileNet system. Work queues are defined on the global level of the FileNet system – they are valid for every process (workflow) definition.
2. Process modelers can define *workflow groups* (*color WG = string*) in every process model. Thus, workflow groups belong to and are valid only in the process (workflow) model in which they are defined. Workflow groups represent teams in FileNet. While executing a task of a process definition, users have the possibility to change the structure of workflow groups of the referring process.

Queues *Work queues* and *personal queues* are two types of queues (*color Q = string*) in FileNet. Queues are pools from which users can select and execute work items. A work queue can have a number of members while a personal queue has only one member. When a work item is offered to a queue, one of the queue members can select and execute the work item. FileNet distributes work in two levels using queues. First, the

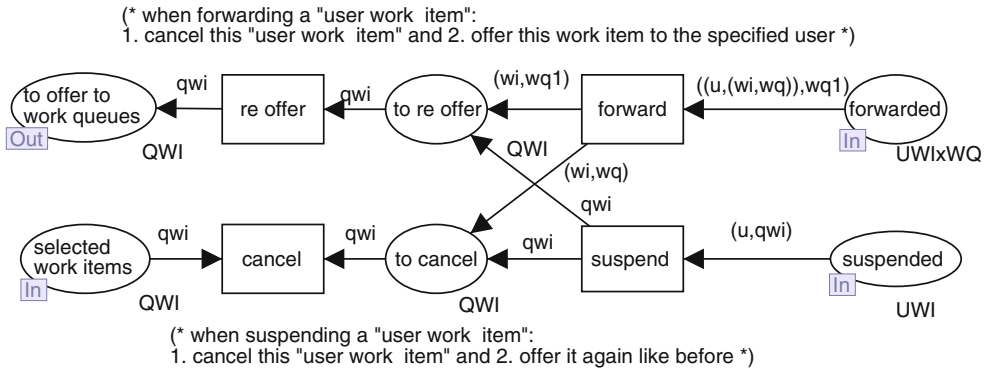

Fig. 11 Staffware—suspend and forward

work item is offered to queues as a *queue work item* (*color QWI = product WI * Q*). Second, the queue work item is offered to the members of the queue as a *user work item* (*color UWI = product User * QWI*).

Figures 12 and 13 show that the model of the two-level work distribution in FileNet is similar to the Staffware model. For more detailed description of this kind of distribution we refer the reader to the Staffware description in Sect. 3.1.1.

Resource allocation FileNet allocates work using work queues and lists of participants. Figure 14 shows that a task in FileNet can be allocated to either a work queue *or* to a list of participants. In this figure we can see that the task "read article" has been allocated to the participants that belong to the workflow group "Information_Systems". Users and workflow groups can be entries of a list of participants. In the FileNet model, task maps are defined as a combination of a task, a list of work groups, and a work queue (*color TMap = product Task * WGs * WQ;*). When defining the input value for a task map, either a work queue or a list of workflow groups should be initiated.

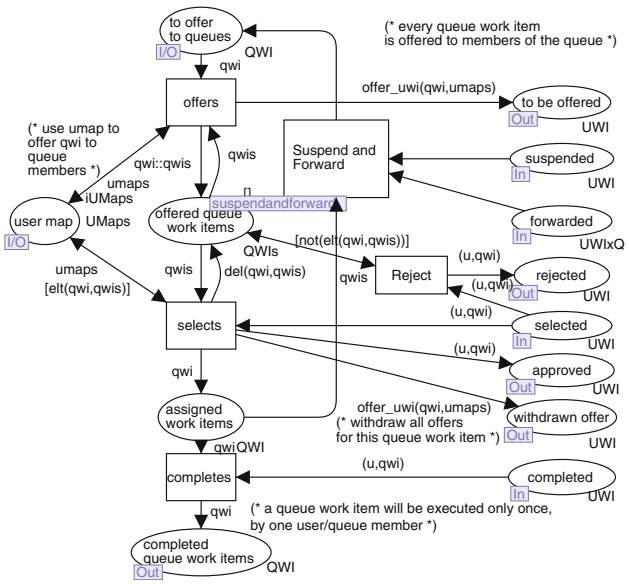

Fig. 13 FileNet—offering

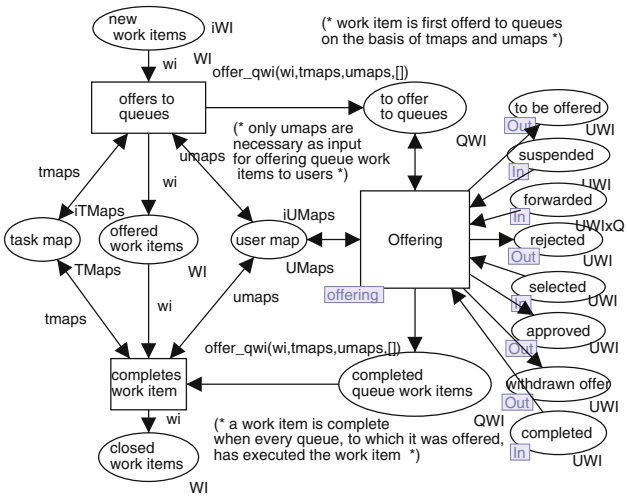

Fig. 12 FileNet—work distribution

If the task is allocated to a work queue, FileNet offers the referring work item to the work queue. If the task is allocated to a list of participants, then it is offered to personal queues of all users that are given as individual participants or are members in participating workflow groups. Allocation via participants is introduced to support team work in FileNet, via the so-called "process voting". During the execution of a task, all participants vote for the specified decision. The work distribution mechanism uses their decisions to determine which work items will be executed next. Since our models abstract from the process perspective, we did not model process voting in the FileNet model.

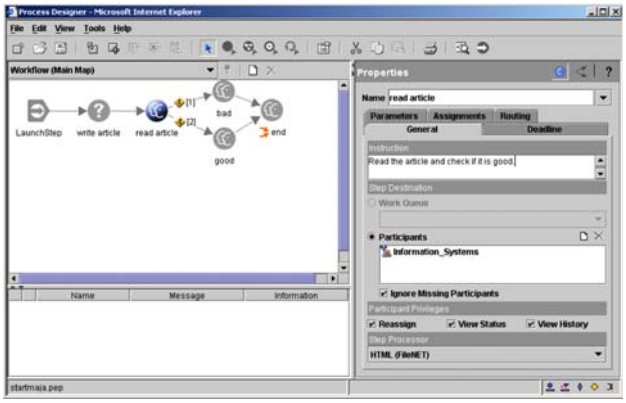

Fig. 14 FileNet—allocation for work queues or participants

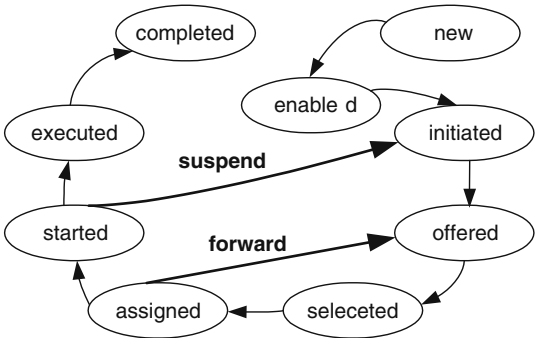

Fig. 15 FileNet—work item life cycle

The allocation function *offer_qwi* allocates queues that are authorized to execute the referring task. Figure 12 shows this function in the inscription on the arc between the transition *offers to queues* and the place *to offer to queues*. This function takes four parameters: (1) the referring work item, (2) task maps, (3) user maps, and (4) an empty list—used as an utility for calculations. This function first searches the task maps for the map of the task that is specified in the work item. The referring task map will point to either a work queue or to a list of participants. In case of a work queue the function produces a queue work item token for the referring work queue. The situation is slightly more complex in the case of a list of participants, because this list may contain users and workflow groups as elements. For each user in the list of participants a queue work item token is produced for the personal queue of the user. For each workflow group in the list of participants queue work items are produced for personal queues of all group members.

Forward and suspend Users can forward and suspend work items in FileNet. When the user selects a work item (s)he can start working on it or forward it to another user. In this case FileNet automatically offers the work item to the new user. When the user is executing a work item s(he) can complete or suspend the work item. In this case FileNet needs to apply the distribution mechanism again, and offer the work item to all allocated users. Figure 15 shows the life cycle of a work item in FileNet. When the life cycle models of FileNet and Staffware (cf. Sect. 3.1.1) are compared, it can be seen that they are identical. Therefore, we use the same adjustments in FileNet like in Staffware models to implement forwarding and suspension: modules *Start Work* and *Stop Work* are changed and sub-module *Suspend and Forward* is added in the Work Distribution module. For detailed description of these sub-modules we refer the reader to Staffware description in Sect. 3.1.1.

3.1.3 FLOWer

FLOWer is a case handling system. Case handling systems differ in their perspective from traditional process-oriented workflow management systems because they focus on the case, instead of the process [3,9]. FLOWer offers a whole case to a user by offering all available work items from the case. When working with FLOWer, the user does not have to follow the predefined order of tasks in the process definition.

To model FLOWer, we extend the Basic Model in such a way that (1) it handles *case-handling distribution* instead of the process-oriented one, (2) it enables the complex *authorization* and *distribution* specifications that FLOWer has, and (3) it enables users to *execute*, *open*, *skip* and *redo* work items.

Case handling To model a case handling system like FLOWer, a number of color sets are introduced. Every process definition in FLOWer is referred to as a case type (*color CaseType = string*). Thus, every case type refers to a list of tasks (*color Tasks = list Task*), which form the process definition (*color Process = product CaseType * Tasks*) for that case type. One case (*color Case = product CaseID* CaseType*) represents an instance of a case type and is identified by a case identification (*color CaseID = INT*).

FLOWer distributes work in two levels. First, a case is distributed to users (*color UCase = product User * Case*). Only one user can select and open the case at one moment. Unlike the distribution in the Basic Model, where distributed work items refer to single tasks, FLOWer distributes whole cases on its first level of distribution. Second, the selected case is opened for the user. Work items (*color WI = product Case * Task*) from the case are offered to the user, based on the

authorization and distribution rules. In the second level of FLOWer distribution users can execute, open, skip and redo work items from the selected case, instead of only executing work items from multiple cases like in the Basic Model, Staffware and FileNet.

Authorization rights Authorization rights are defined for every case type. First, process-specific roles are defined within a case type (*color PRole = product Role * CaseType*). Second, to make authorization rights, roles are assigned to tasks within the case type. These authorization rights are stored in task maps (*color TMap = product Task * Role * CaseType*). The authorization rights determine what users *can do* and are applied by the distribution mechanism when opening the case for the user. The user is allowed to work only on tasks for which (s)he has the authorized roles. Although authorization exists in the Basic Model, Staffware and FileNet, in these models it is defined on the global (system) level, instead of embedding roles in process models. Rather, roles are defined in the global organizational model.

Distribution rights Distribution rights define what users *should do*. Unlike authorization rights, distribution rights are defined on the global level of the FLOWer system, and are valid for all case types. These rights can be used to model the organizational structure and to assign authorization rights from the process definitions (case types) to users. *Function profiles* and *work profiles* define distribution rights. Function profile has a unique function name (*color FN = string*) and a list of case type authorization roles (*color FP = product FN * PRoles*). If, for example, there are two case types (two processes) – one with "secretary1" and the other with "secretary2" as an authorization role, the function profile "secretary" could include both authorization roles. When we would assign the function profile "secretary" to a user, we would indirectly assign both authorization roles from two processes. Work profiles assign function profile(s) to users and they can be used to structure organization into groups, departments or units. One Work Profile consists of a unique name (*color WN = string*), a list of users and a list of function profiles (*color WP = product WN * Users * FNs*). Distribution rights are used to define the organizational model in FLOWer. While this model is independent of the authorizations in the Basic Model, Staffware and FileNet, in FLOWer it has to be related to special authorization roles form case types. In this way, FLOWer creates two-layered organization specification: one part of it is in the distribution rights and the other in the authorization rights.

Distribution of cases Figure 16 shows the Work Distribution module of FLOWer. In this module, FLOWer model distributes new cases to users, instead of distribution work items like the Basic Model. When a new case token is available in the place *new cases*, the transition *offers case* fires. This transition consumes the case token from the place *new case* and retrieves task maps, work profiles and function profiles from places *task map*, *work profile* and *function profile*, respectively. It adds a token to the list of cases in the place *offered cases*. This place stores a list of cases that were offered to users but not yet selected by any user. The most important effect of this transition is that it produces a user case token in the place *offer case*, via the case allocation function *offerc* in the arc inscription. A user case token in the place offer cases sends a message to the Work Lists module to offer the referring case to the referring user. The allocation function *offerc* takes four parameters: (1) case that will be allocated, (2) task maps to find the mapping of the referring case type task to the case type role, (3) a list of function profiles to find the ones that contain the case-type-specific role from the task maps, and (4) a list of work profiles to find the users that are assigned to the appropriate function profiles. Next, the Work Distribution module waits for the message from the Work Lists module that a user wants to select a case.

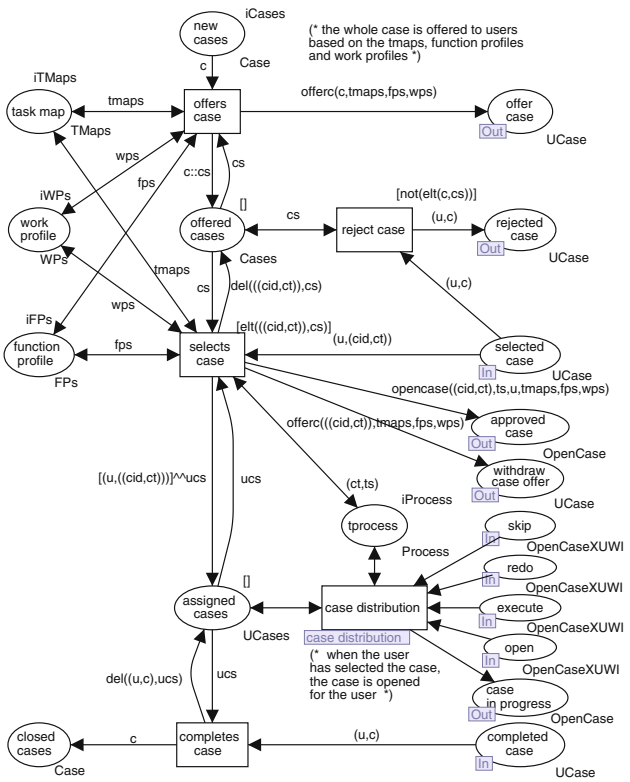

Fig. 16 FLOWer—work distribution

This message arrives with a user case token in the place *selected case*. Only one user can select a case at the same time in FLOWer. The Work Distribution module responds to this message accordingly to this rule by checking if the case has already been selected, i.e., if the referring case is contained in the list of offered cases in the place *offered cases*. Transitions *selects case* and *reject case* are the two alternative transitions that can respond to the new user case token in the place *selected case*. The transition *selects case* will fire if the referring case is contained in the list of cases in the place *offered cases*, which can be seen in the guard of the transition. This transition will consume the user case token from the place *selected case*, remove the referring case from the list of cases in the place *offered cases* and produce the user case token in the place *assigned cases*. By removing the token from the place *offered case*, we assure that the case cannot be selected again. The transition *selects case* also sends two messages to the Work Lists module. First, since a user can select the case, a message is sent to the Work Lists module to withdraw all offers of the referring case. The transition *selects case* sends this message by producing all previous offers of the referring case in the place *withdraw case offer*. Second, the approval message for the selection of the case (for the user) is sent by producing the referring open case token in the place *approved case*.

The function *opencase* in the arc inscription between the transition *selects case* and the place *approved case* produces an open case token. This function takes six parameters: (1) the identification and the type of the case to be open, (2) the tasks that are contained in the process definition of the case type, (3) the user for whom the case is open, (4) a list of task maps to find case type authorized roles for every task, (5) a list of function profiles to search for the ones that contain the authorization roles for the tasks, and (6) a list of work profiles to determine which of the selected function profiles are assigned to the user. The open case token (*color OpenCase = product UCase*CaseState*) that is produced stores the information about the user, case and the state of the case (*color CaseState = product WIs*WIs*WIs*WIs*). The case state consists of four lists of work items that are: (1) *waiting* to be enabled, (2) *active* (i.e. they are enabled and can be executed), (3) *finished* (executed), and (4) *skipped*. When the case is opened for the first time, the list of active items contains the first work item, the list of waiting items all the other authorized work items, and the lists of executed and skipped items are empty.

After the Work Distribution module opens the case for the user, the Case Distribution sub-module handles the distribution within the case. This sub-module manages events when users work on tasks within the case. We refer to this part of the FLOWer work distribution as to the distribution within the case and describe it further in this paper.

The last message that arrives from the Work Lists module is that the user has finished working with the case. This message arrives with a new user case token in the place *completed case*. The transition *completes case* consumes this token, removes the referring user case token from the list of assigned cases in the place *assigned cases* and produces the referring case token in the place *closed*. Although, in FLOWer system, after the case has been closed it is possible to be offered again, we do not model this in the FLOWer CPN model due to the complexity and size of the model. However, it is possible to include this behavior in the model by: (1) returning the closed case token to the place new case, and (2) storing permanently the state of every case, similarly like task maps, function profiles, work profiles and process definitions.

Figure 17 shows the Work Lists module of the FLOWer model. Generally, the functionality of the part of this module that deals with the distribution of cases is the same as the Work Lists module of the Basic Model. However, there are some differences between

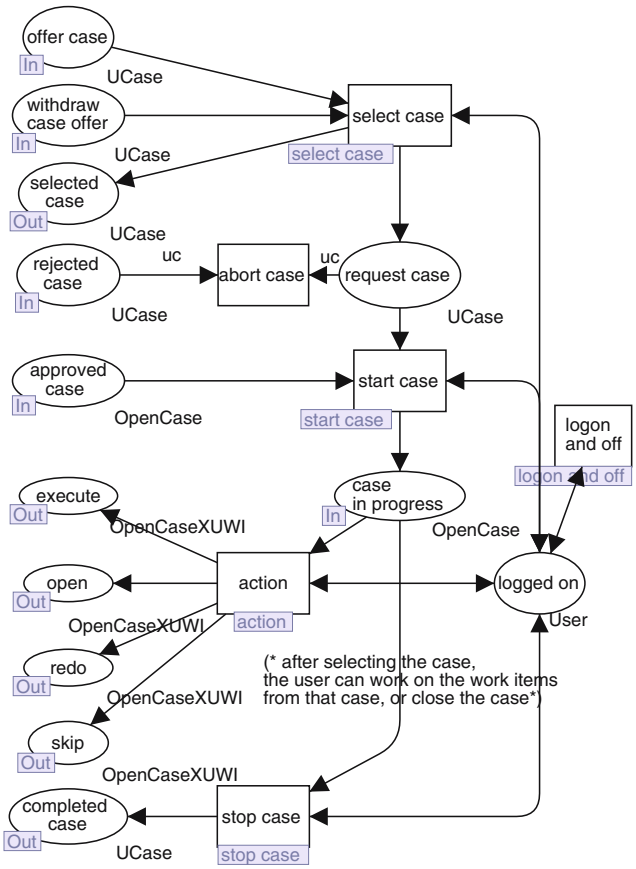

Fig. 17 FLOWer—work lists

these two modules. First, the places are named differently to match the context. There are two kinds of places in the FLOWer model: (1) names of the places and transitions that deal with the case distribution contain the word "case" (e.g., the place *offer case*), and (2) names of the places that deal with the distribution within the case do not contain the word "case" (e.g., place *execute*). Second, the places that deal with the distribution are of the user case type, instead of the user work item type. Finally, the sub-module Action deals with actions of users in the context of the distribution within a case. The Action sub-module is described further in this paper, within the paragraph about the distribution within the case. Because the distribution of the cases in the FLOWer Work Lists module is similar to the distribution of work items in the Work Lists module of the Basic Model, for a detailed description we refer the reader to Sect. 2.

Distribution within a case When working with traditional, process-oriented, systems users can mostly execute or cancel work items. This property of such systems can be found in the Basic Model, Staffware model and FileNet model. Unlike these process-oriented systems (models), a case handling system FLOWer, and its model, allow users to perform *four actions* on work items: open, execute, skip and redo. Figure 18 shows that the life cycle of a work item in FLOWer is somewhat more complex that the life cycles of the other models. Because a user selects a whole case, work items are *assigned* to the user before they are *enabled*. Following the process definition of a case type (because of the complexity we assume this to be a sequence of tasks) the FLOWer systems *enables* the next work item in the sequence. After the user *selects* an enabled work item, (s)he starts with its execution and the work item is transferred to the state *execute*. Once the execution stops, the work item becomes *completed*. It is possible to *skip* an *enabled* work item and transfer it without the execution to the state *completed*. Besides *enabled* items, the user can also skip work items that are *assigned*. The user can *open* and start an *assigned* work item. By redoing a *completed* work item, the user transfers the work item to the state *enabled*.

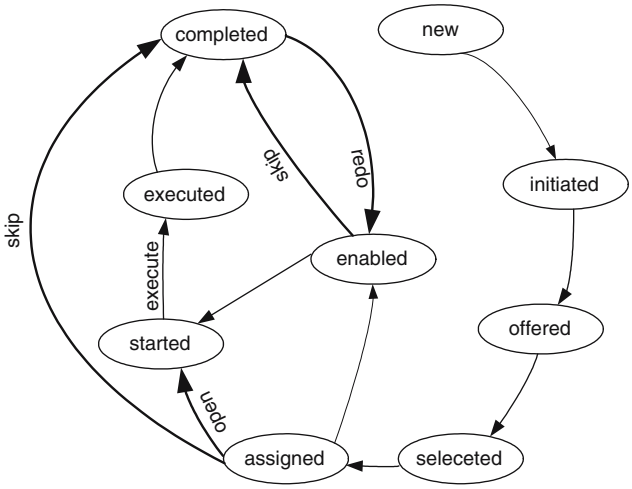

Fig. 18 FLOWer—work item life cycle

The state of the case plays an important role in the distribution within the case from two perspectives: First, which of the four actions are possible depends on the state of the case. For example, it is only possible to execute the work items that are contained in the list of active work items in the case state. Second, each of the four actions changes the state of the case. For example, if a work items was executed, it is removed from the list of active items and added to the list of finished items in the case state. We use Table 5 to explain the role of the case state in the distribution within the case. Each of the four rows refers to one of the four actions—the name of the action is stated in the column "action". The first column ("work item was") is a precondition that states from which list in the case state the work item has to be selected in order to undergo the referring action (e.g., the action open can be applied only to waiting work items). The column "work item becomes" is a postcondition that states to which list the selected work item will be moved after the action (e.g., after opening, the work item is moved to the list of active items in the case state). Finally, the postcondition column "side effects"

Table 5 FLOWer—the four actions

Preconditions	Action	Postcondition	
Work item was		Work item becomes	Side effects
Waiting	Open	*Active*	*Waiting* and *active* items that succeed become *skipped*.
Active	Execute	*Finished*	The direct successor becomes *active*.
Active or *waiting*	Skip	*skipped*	Succeeding *waiting* and *active* items become *skipped*. The direct predecessor becomes *active*.
Finished or *skipped*	Redo	*Active*	Preceding *finished* and *skipped* items become *waiting*.

states what are the possible side effects of the action. For example, when opening an work item, other (if any) waiting and/or active items after the opened item are moved to the list of skipped items in the case state.

When a user selects a case, FLOWer continues work distribution within that case—the work items of the case are distributed to the user. Before this distribution can start, the Work Distribution module (cf. Fig. 16) opens the case by sending the appropriate message to the Work Lists module (cf. Fig. 17) with an open case token in the place *approved case*. After the Work Lists module receives this message, its transition *start case* automatically fires by consuming the open case token from the place *approved case* and producing one in place *case in progress*. Once the open case token is produced in place *case in progress*, the distribution within the case starts and the user can work on the work items in that case.

Figure 19 shows the Action sub-module, which is a new sub-module in the FLOWer Work Lists module. This sub-module handles the actions of a user when s(he) works within a case and makes sure that the *preconditions* (cf. Table 5) are met before each of the four actions can take place. The Action sub-module can be seen as an extension of the Start Work sub-module of the Basic Model. In the Start Work sub-module the user can only start the work item in progress. However, when an open case is in progress in the Action sub-module the user can: (1) *execute* the work item which is next in the process definition of the case type — an item contained in the list of active items in the case state; (2) *open* for executing a work item that is still not ready for execution according to the process definition of the case type —an item contained in the list of waiting items in the case state; (3) *skip* a work item that is currently enabled or waiting to be enabled—an item contained in the lists of active or waiting items in the case state, or (4) *redo* a work item and execute again a work item which has already been executed – an item contained in the lists of finished or skipped items in the case state. Four transitions in the Action sub-module refer to the four actions of users—open, execute, skip, and redo. All transitions retrieve an user token from the place *logged on*, to make sure that only the users who are currently logged on can perform these actions. Also, all transitions consume the open case token from the place *case in progress*. The open case token stores the information about the user, case, and case state (i.e., lists of waiting, active, finished and skipped work items for that case). It is necessary to consume(remove) the open case token from the place *case in progress* because after every action, the Work Distribution module (more specifically — its Case Distribution sub-module) changes the state of the case, which is stored in the open case token in the place *case in progress*. After performing one action, the user cannot perform the next action before the Case Distribution sub-module updates the case state and produces the referring open case token in the place *case in progress*. The transition *open* can fire only if the list of waiting items in the case state is not empty, as can be seen in the guard of this transition. The transition *open* produces a token in the place *open*. This place is of a complex

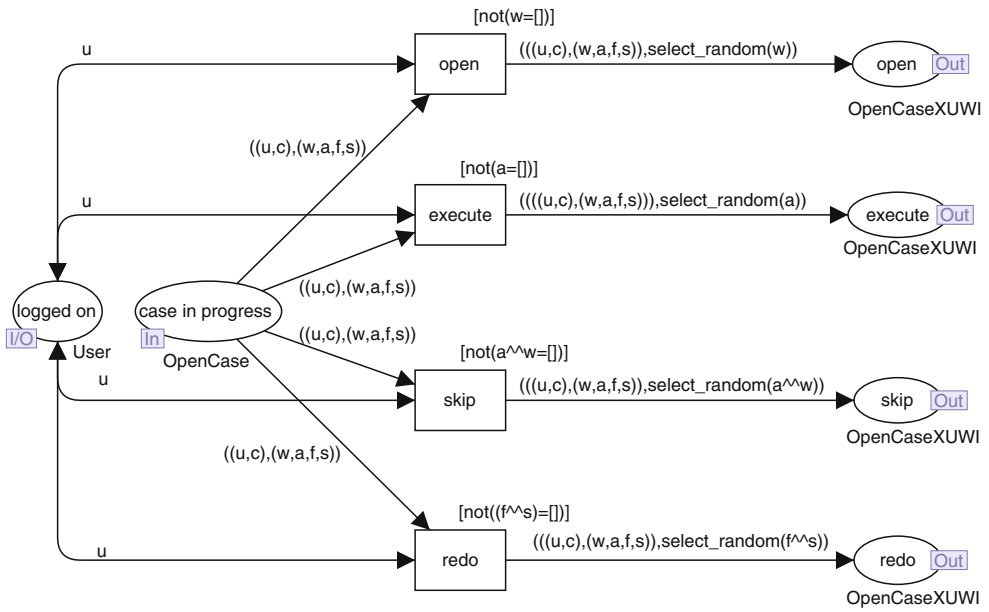

Fig. 19 FLOWer—action

type, which consist of an open case and an user work item. When a token is produced in this place, the message is sent to the Case Distribution sub-module that the referring user work item should be open in the referring open case. Although a user who works with FLOWer can freely choose which item should be open, for the simplicity we use an random function to select an item from the list of waiting items. The inscription on the arc from the transition *open* produces a token from the current open case and the (randomly) selected waiting item in the place *open*. Similarly, according to the preconditions (cf. Table 5), guards on transitions *execute*, *skip*, and *redo* ensure that they fire only when lists of *active*, *active and waiting*, and *finished and skipped* items are not empty, respectively. Places execute, skip and redo are of the same type as the place open, and a token in each of those places sends a message to the Case Distribution module that the referring user work item should be executed, skipped or redone, respectively. Following the preconditions (cf. Table 5), the inscriptions on the arcs between the (1) transition and place *execute*, (2) transition and place *skip* and (3) transition and place *redo* each create a token containing the open case and the randomly selected (1) active, (2) active or waiting, and (3) finished or skipped work item in the places (1) *execute*, (2) *skip* and (3) *redo*, respectively.

When working on a case in the FLOWer system, users work with the interface tool "Wave Front" [41] where they can see the state of the open case. Users can see which work items are waiting, active, finished and skipped. Figure 20 shows one example of an open case in the "Wave Front". The first two tasks ("Claim Start" and "Register Claim") are *finished* work items and they are marked with a 'check' symbol. The third work item ("Get Medical Report") was *skipped*, as can be seen from the 'arrow' symbol. Thus, finished and skipped work items are presented after the "Wave Front" line. The three *active* work items on the Wave Front line are "Get Police Report", "Assign Loss Adjuster" and "Witness Statements". Finally, the two last work items ("Policy Holder Liable" and "Close Case") are *waiting* before the Wave Front line to become active.

Fig. 20 FLOWer wave front

Case Distribution is a sub-module of the FLOWer Work Distribution module. This sub-module responds to user's requests to open, execute, skip or redo work items in the distribution within the case. The task of the Case Distribution sub-module is to respond to the actions of users by changing the state of the case accordingly to the postcondition of every action (cf. Table 5). Figure 21 shows the Case Distribution sub-module of the FLOWer model. The requests (messages) for actions are received via tokens in places *open*, *execute*, *skip* and *redo*. These places are of the type which stores the information about the open case (the user case and the case state) and the user work item to which the action (open, execute, skip or redo) should be applied. Due to delays, it is possible that a message to execute a work item from the case arrives after the case had been closed, the transition *ignore* behaves as a "garbage collector" of such requests. This transition retrieves user case token from the place *assigned cases* and consumes a token from the places *open*, *execute*, *skip* and *redo*. Thus, when the transition *ignore* fires, the message to perform an action is ignored and removed from the model. The guard on this transition makes sure that the transition will fire only if the case is not closed, i.e., the appropriate user case token is not found in the list in place *assigned cases*. Transitions *open*, *execute*, *skip* and *redo* fire when tokens arrive to places *open*, *execute*, *skip* and *redo*, respectively. Each of these transitions consumes the arrived token from the appropriate place (e.g., transition *open* consumes the arrived token from the place *open*) and retrieves the list of user cases from the place *assigned cases*. Guards on transitions *open*, *execute*, *skip* and *redo* show that they will fire if the request is valid, i.e., if the appropriate user case token is found in the list in the place *assigned cases*. The result of each of those four transitions is a produced open case token in the place *case in progress*. The inscriptions on the arcs between these transitions and the place *case in progress* change state of the case, accordingly to the postcondition of each action. More specifically, the new case state is created by four functions in the inscriptions on the arcs between transitions *open*, *execute*, *skip* and *redo*, and the place *case in progress*. These functions take three parameters: (1) user work item to which the action should be applied, (2) the old case state that should be changed, and (3) the case process definition – the tasks of the case type. The third parameter is retrieved from the place *tprocess*, which stores process definitions for all case types. Functions *open_item*, *execute_item*, *skip_item* and *redo_item* create the new case state accordingly to the postcondition

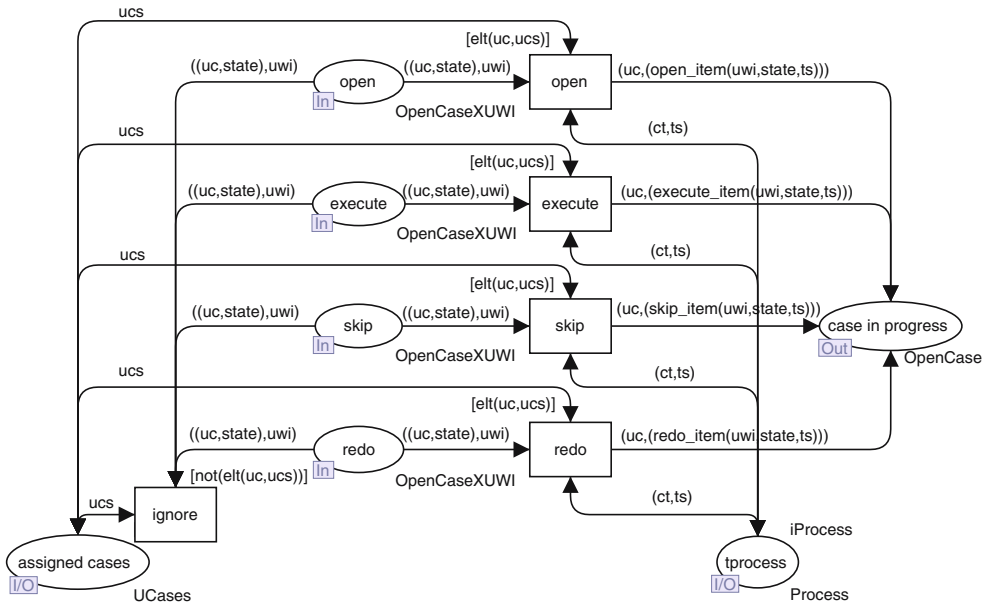

Fig. 21 FLOWer—case distribution

of the referring action (cf. Table 5). When a token is produced in the place *case in progress*, a message is sent to the Action sub-module that the user can select the next action for the referring open case.

The FLOWer CPN model implemented significant changes to the Basic Model. Because of its case-handling nature, the FLOWer model differs the most from the other CPN models. The greatest differences are caused by the fact that the system distributes the cases and the work items within the cases, instead of only work items. The Start Work module of the Basic Model was significantly extended because users can open, execute, skip and redo work items in FLOWer. Regardless the differences between FLOWer and process-oriented systems (modelled by the Basic Model, Staffware model and FileNet model), it was possible to extend the Basic Model to the FLOWer work distribution model.

3.2 Resource patterns

Instead of extending the Basic Model for more systems, we also looked at a more systematic way of work distribution. As indicated, similar concepts are often named and presented differently in different workflow management systems. Therefore, it is interesting to define these concepts in a system-independent manner. We have used 43 documented *resource patterns* [46,48]. These patterns can be used as representative examples for analyzing, evaluating and comparing different workflow management systems with respect to work distribution. Resource patterns are grouped into a number of categories: *creation patterns*, *push patterns*, *pull patterns*, *detour patterns*, *auto-start patterns*, *visibility patterns*, and *multiple resource patterns*. Each of these patterns can be modelled in terms of a CPN model.

Table 6 shows an overview of the patterns. It also shows whether a pattern is directly supported by the three systems (SW = Staffware, FN = FileNet, FW = FLOWer) and the Basic Model (BM). The Basic Model supports less patterns than any of the three systems. This makes sense since each of the system-specific models can be seen as an extension of the Basic Model. It is interesting to see that existing systems typically support less than half of the patterns directly. This reveals typical limitations of contemporary products. Some of the patterns are considered out-of-scope for our models (marked with "o"). These are typically patterns directly depending on control-flow functionality, while we prefer to focus exclusively on work distribution. Each of the patterns not marked with "o" can easily be added to the Basic Model separately.

We cannot elaborate on each of the patterns, but we will discuss four to illustrate our work. None of the systems supports *Pattern 16: Round Robin*, *Pattern 17: Shortest Queue*, *Pattern 38: Piled Execution*, and *Pattern 39: Chained Execution*. Patterns 16 and 17 are push patterns, i.e., a patterns to push work to a specific user. As auto-start patterns, patterns 38 and 39 enable the automatic start of the execution of the next work item once the previous has been completed.

Table 6 Support for resource patterns in three workflow systems and basic model

Nr	Pattern	SW	FN	FW	BM
1	Direct allocation	+	+	+	+/–
2	Role-based allocation	+	+/–	+	+
3	Deferred allocation	+	+	–	–
4	Authorization	–	–	+	–
5	Separation of duties	–	–	+	–
6	Case handling	–	–	+	–
7	Retain familiar	–	–	+	–
8	Capability-based allocation	–	–	+	–
9	History-based allocation	–	–	–	–
10	Organizational allocation	+/–	+/–	+/–	+/–
11	Automatic execution	+	+	+	o
12	Distribution by offer—single resource	–	–	–	–
13	Distribution by offer—multiple resources	+	+	+	+
14	Distribution by allocation—single resource	+	+	+	–
15	Random allocation	–	–	–	+
16	Round robin allocation	–	–	–	–
17	Shortest queue	–	–	–	–
18	Early distribution	–	–	+	–
19	Distribution on enablement	+	+	+	+
20	Late distribution	–	–	–	–
21	Resource-initiated allocation	–	–	+	+
22	Resource-initiated execution—allocated work item	+	+	+	+
23	Resource-initiated execution—offered work item	+	+	–	–
24	System-determined work list management	+	+	+	o
25	Resource-determined work list management	+	+	+	o
26	Selection autonomy	+	+	+	+
27	Delegation	+	+	–	–
28	Escalation	+	+	–	–
29	Deallocation	–	–	–	–
30	Stateful reallocation	+/–	+	–	–
31	Stateless reallocation	–	–	–	–
32	Suspension/resumption	+/–	+/–	–	–
33	Skip	–	–	+	o
34	Redo	–	–	+	o
35	Pre-Do	–	–	+	o
36	Commencement on creation	–	–	–	–
37	Commencement on allocation	–	–	–	–
38	Piled execution	–	–	–	–
39	Chained execution	–	–	+	–
40	Configurable unallocated work item visibility	–	–	–	o
41	Configurable allocated work item visibility	–	–	+	o
42	Simultaneous execution	+	+	+/–	+
43	Additional resources	–	–	–	–

+ = direct support, – = no direct support, +/– = partial support, o = out-of-scope

Round robin and shortest queue Round Robin and Shortest Queue push the work item to one user of all users that qualify. Round Robin allocates work on a cyclic basis and Shortest Queue to the user with the shortest queue. This implies that each user has a counter to: (1) count the sequence of allocations in Round Robin and (2) count the number of pending work items in Shortest Queue.

As Figs. 22 and 23 show, these two patterns are implemented in a similar way in the Work Distribution Module. The required changes to the Basic Model are minimal. A counter is introduced for each user (token in the place *available*) and functions *round_robin* and *shortest_queue* are used to select one user from the set of possible users based on these counters. These allocation function are used in the inscription on the arc(s) between the transition *offers* and the place *to allocate*. Both functions take two parameters: (1) the set of the "classical" allocation created by the allocation function *offer* from the Basic Model, and (2) appropriate counters. Both functions allocate the right user work item via three steps: (1) take the set of user work items created by the

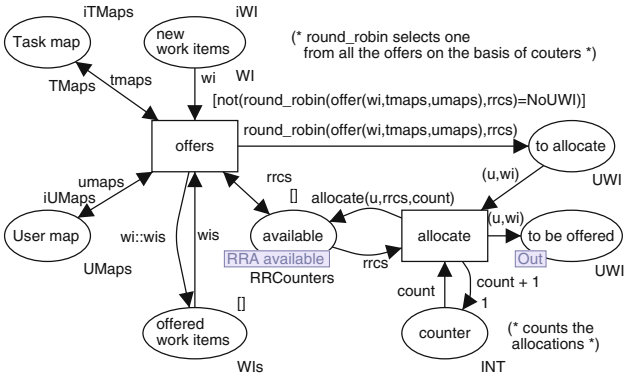

Fig. 22 Push patterns—round robin

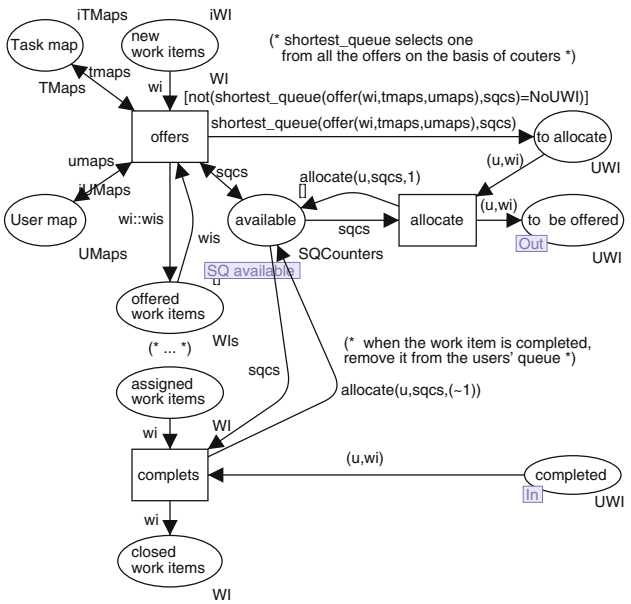

Fig. 23 Push patterns—shortest queue

allocation function *offer*; (2) for every user work item search for the value of the counter; and (3) select and return only the user work item where the user has the smallest value of the counter. In this way, push allocation functions can be seen as a filter that selects only one out of the set of allocations. The model for Shortest Queue has an additional connection (the two arcs between the transition *complete* and the place *available*) that updates the counter when a work item is completed to remove it from the queue (decrease the value of the counter for the referring user).

Piled and chained execution Piled and Chained Execution are auto-start patterns, i.e., when the user completes execution of current work item the next work item starts automatically. When working in Chained Execution, the next work item will be for the same *case* as the completed one—the user works on different tasks for one case. Similarly, if the user works in Piled Execution the next work item will be for the same *task* as the completed one – the user works on one task for different cases. Figures 24 and 25 show that Piled and Chained Execution are implemented similarly in the Stop Work sub-module. Users can choose to work in the normal mode or in the auto-start mode (which is represented by the token in place *special mode*). The function *select* is implemented to search for the next work item for

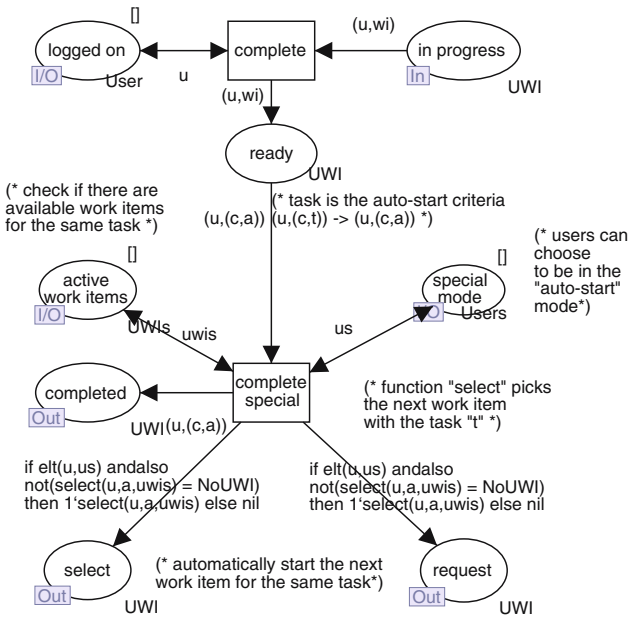

Fig. 24 Piled execution—stop work

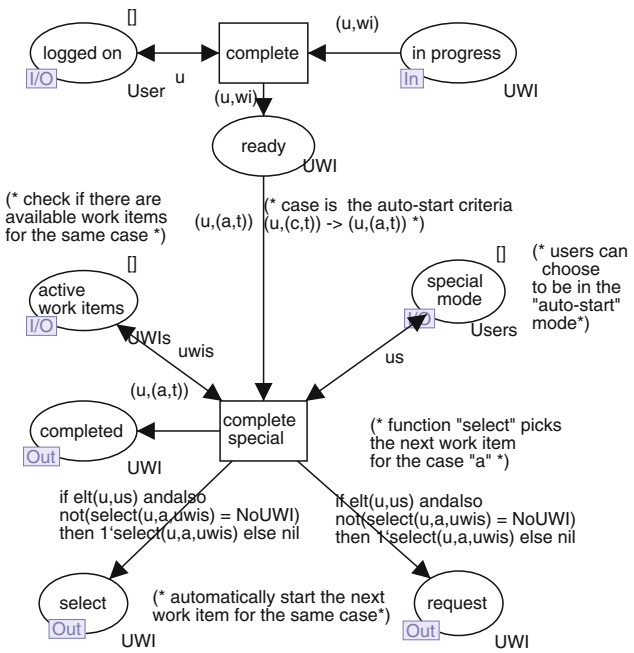

Fig. 25 Chained execution—stop work

the same: (1) *task* in Piled Execution and (2) *case* in Chained Execution. The different auto-start criteria are passed in the inscription of the arc between the place *ready* and transition *complete special*. In the case of Piled Execution, the auto-start criteria is the task, and in the case of Chained Execution the auto-start criteria is the case. These two models show that the transition *complete special*, besides the usual connection to the place *completes*, has the connections of a "start work" transition: it retrieves items from the place *active work items*, and produces items in the places *request* and *select*. The inscriptions on arcs leading to places *request* and *select* first check if the user is working in the special (auto-start) mode and a user work item is available according to the auto-start criteria. If these conditions are fulfilled, the next user work item is auto-started, i.e., an appropriate user work item token is produced in the places *request* and *select*. The function *select* is implemented to search for the next work item for the same: (1) task in Piled Execution and (2) case in Chained execution.

4 Related work

Since the early nineties workflow technology has matured [24] and several textbooks have been published, e.g., [5,19,28,35,40]. During this period many languages for modelling workflows have been proposed, i.e., languages ranging from generic Petri-net-based languages to tailor-made domain-specific languages. The Workflow Management Coalition (WfMC) has tried to standardize workflow languages since 1994 but failed to do so [23]. XPDL, the language proposed by the WfMC, has semantic problems [2] and is rarely used. In a way BPEL [11] succeeded in doing what the WfMC was aiming at. However, both BPEL and XPDL focus on the control-flow rather than the resource perspective.

Despite the central role that resources play in workflow management systems, there is a surprisingly small body of research into resource and organizational modelling in the workflow context [1,33]. In early work, Bussler and Jablonski [15] identified a number of shortcomings of workflow management systems when modelling organizational and policy issues. In subsequent work [28], they presented one of the first broad attempts to model the various perspectives of workflow management systems in an integrated manner including detailed consideration of the organizational/resource view.

One line of research into resource modelling and enactment in a workflow context has focused on the characterization of resource managers that can manage organizational resources and enforce resource policies. In [18], the design of a resource manager is presented for a workflow management system. This work includes a high level resource model together with proposals for resource definition, query and policy languages. Similarly, in [34], an abstract resource model is presented in the context of a workflow management system although the focus is more on the efficient management of resources in a workflow context than the specific ways in which work is allocated to them. In [27], a proposal is presented for handling resource policies in a workflow context. Three types of policy—qualification, requirement and substitution—are described together with a means for efficiently implementing them when allocating resources to activities.

Another area of investigation has been into ensuring that only appropriate users are selected to execute a given work item. The Role-based access control (RBAC) model [21] presents an approach for doing this. RBAC models are effective but they tend to focus on security considerations and neglect other organizational aspects such as resource availability.

Several researchers have developed meta-models, i.e., object models describing the relation between workflow concepts, which include work allocation aspects, cf. [8, 38–40,45]. However, these meta-models tend to focus on the structural description of resource properties and typically do not describe the dynamics aspects of work distribution.

Flexibility has been a research topic in workflow literature since the late nineties [4,7,9,10,16,20,26,31,42,44, 52]. Flexibility triggers all kinds of interesting research questions, e.g., if a process changes how this should influence the running cases [7]? Examples of qualitative analysis of flexibility of workflow management system can be found in [13] and [25]. One way of allowing for more flexibility is to use the case handling concept as defined in [3,9]. FLOWer [12,41] can be seen as a reference implementation of the case handling concept. Therefore, its resource perspective was modelled in this paper. Besides FLOWer there are few other case handling tools: E.C.H.O. (Electronic Case-Handling for Offices), a predecessor of FLOWer, the Staffware Case Handler [50] and the COSA Activity Manager [49], both based on the generic solution of BPi [14], Vectus [36, 37], and the open-source system con:cern (http://www.con-cern.org/).

The work reported in this paper can be seen as an extension of the *workflow patterns initiative*[4]. Besides a variety of control-flow [6] and data [47] patterns, 43 resource patterns [46,48] have been defined. This paper complements the resource patterns [46,48] by providing executable models for work distribution mechanisms.

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

5 Discussion

Workflow management systems should provide flexible work distribution mechanisms for users. This will increase the work satisfaction of users and improve their ability to deal with unpredictable situations at work. Therefore, work distribution is investigated as the functionality provided for the user—workflow management systems are tested in laboratories [46,48] or observed (in empirical research) in companies [13]. This kind of research observes systems *externally* and provides insights into *what* systems do. Analysis of the systems from an *internal* perspective can explain *how* systems provide for different work distribution mechanisms. Due to the complexity of workflow management systems as software products, internal analysis starts with developing a model of the system. Unlike the mostly used static models (e.g., UML class diagrams, entity-relationship diagrams), dynamic models (e.g., CPN models) provide for interactive investigation of work distribution as a dynamic feature. CPN models can be used for the investigation of both *what* systems do and *how* they do it.

Workflow management systems often provide for different features or use different naming for the same features. Investigation of work distribution requires analysis, evaluation and comparison of models of several systems. In order for models of different systems to be comparable, it is necessary to start with developing a common framework – a *reference model*. We have developed the Basic Model as a reference model for work distribution mechanisms in workflow management system. The models of Staffware, FileNet, FLOWer and resource patterns are comparable because all models are developed as extensions of a reference model—the Basic Model.

The model of a workflow system is structured into two modules. The Work Distribution module represents the core of the system which is often called the "workflow engine". The Work Lists module represents the so-called "work list handler" of a workflow system and it serves as an interface between the workflow engine and users. The interface between the two modules (i.e., the messages that are exchanged between them) should contain as little information as possible about the way work items are managed in modules. The Work Lists module should abstract from the way the work items are created, allocated and offered in the Work Distribution module. The reverse also holds: how work items are actually processed by users is implemented in the Work Lists module. Once a proper interface is defined, it is easy to implement various ways of work distribution by adding/removing simple features in either one of the modules. For example, push patterns (Round Robin and Shortest Queue) are implemented in the Work Distribution module and auto-start resource patterns (Chained and Piled Execution) in the Work Lists module.

Work distribution mechanism determines what users can do with work items. In the Basic Module the user follows a fixed predefined path by only executing work items. Users of Staffware and FileNet models have the freedom to forward and suspend work. In FLOWer, as the most flexible system, users have four possibilities: execute, open, skip and redo work. Our models show that a more complex model work distribution adds messages between the Work Distribution and Work Lists modules. These new messages correspond to new actions (operations) that users can do.

Both the system-based and the patterns-based CPN models showed that one of the core elements of work distribution is the "allocation algorithm". This algorithm includes the "rules" for work distribution. It is implemented in the Work Distribution module as the function *offer*, which allocates work based on (1) new work items, (2) process definition, and the (3) organizational model. This function should be analyzed further in order to discover an advanced allocation algorithm, which should be more configurable and less system-dependent.

Every system has its own method of modelling organizational structure. Staffware models groups and roles. In FileNet the organizational model includes groups of users and teams, but does not model roles. FLOWer groups users based on a hierarchy of roles, function profiles and work profiles. Thus, each of the system offers a unique predefined type of the organizational structure. Since every allocation mechanism uses elements of the organizational model, limitations of the organizational model can have a negative impact on the work distribution in the system. For example, because in Staffware one role can be assigned to only one user, it is not be possible to offer a work item to a set of "call center operator"-s.

Each of the three models of workflow management systems distributes work using two hierarchy levels. Staff-ware and FileNet use two levels of work distribution: queue work items are first distributed to work queues, and then work items are distributed within each of the work queues. The FLOWer model starts with the case distribution and then distributes work items of the whole case. Although all three systems distribute work at two levels, they have unique *distribution algorithms* (the set of allocation rules implemented in the function *offer*) and *objects of distribution* (work items, queue work items, cases).

Models of resource patterns [46,48] show that push patterns (Round Robin and Shortest Queue) can be

implemented "on top of" the pull mechanism, as a filter. Once the pull mechanism determines the set of allocated users, the "push" allocation function extracts only one user from this set. Auto-start patterns turned out to be remarkable straightforward to model, triggering the question why this is not supported by systems like Staffware and FileNet (FLOWer supports the Chained Execution in a limited form).

6 Conclusions

This paper focused on the *resource perspective*, i.e., the way workflow management systems distribute work based on the structure of the organization and capabilities (qualifications) of people. To understand work distribution, we used the CPN language and CPN Tools to model and analyze different work distribution mechanisms. To serve as a 'common basis' model, we provided a model that can be seen as a reference model of existing workflow management systems. This model was extended for models of three workflow management systems—Staffware, FileNet, and FLOWer. Although the reference model already captures many of the *resource patterns*, we also modelled four more advanced patterns by extending the reference model. In contrast to existing research that mainly uses static models (e.g., UML class diagrams), we focused on the dynamics of work distribution. Our experiences revealed that it is relatively easy to model and analyze the workflow systems and resource patterns using CPN Tools. This suggests that CPN language and the basic CPN model are a good basis for future research. We plan to test completely new ways of work distribution using the approach presented in this paper. The goal is to design and implement distribution mechanisms that overcome the limitations of existing systems.

References

1. van der Aalst, W.M.P.: Don't go with the flow: Web services composition standards exposed. IEEE Intell. Syst. **18**(1), 72–76 (2003)
2. van der Aalst, W.M.P.: Business process management demystified: a tutorial on models, systems and standards for workflow management. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets, Lecture Notes in Computer Science, vol. 3098, pp. 1–65. Springer-Verlag, Berlin, (2004)
3. van der Aalst, W.M.P., Berens, P.J.S.: Beyond workflow management: product-driven case handling. In: Ellis, S., Rodden, T., Zigurs, I. (eds.) International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001), pp. 42–51. ACM Press, New York (2001)
4. van der Aalst, W.M.P., Desel, J., Oberweis, A.: (eds.) Business Process Management: Models, Techniques, and Empirical Studies. Lecture Notes in Computer Science, vol. 1806 Springer Berlin (2000)
5. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT press, Cambridge (2002)
6. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)
7. van der Aalst, W.M.P., Jablonski, S.: Dealing with workflow change: identification of issues and solutions. Int. J. Comput. Syst. Sci. Eng. **15**(5), 267–276 (2000)
8. van der Aalst, W.M.P., Kumar, A.: Team-enabled workflow management systems. Data Knowl. Eng. **38**(3), 335–363 (2001)
9. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2), 129–162 (2005)
10. Agostini A., De Michelis G.: Improving flexibility of workflow management systems. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management: Models, Techniques, and Empirical Studies. Lecture Notes in Computer Science, vol. 1806, pp. 218–234. Springer, Berlin (2000)
11. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana S.: Business process execution language for Web services, Version 1.1. Standards proposal by BEA Systems. International Business Machines Corporation, and Microsoft Corporation (2003)
12. Pallas Athena. Case Handling with FLOWer: Beyond workflow. Pallas Athena BV, Apeldoorn (2002)
13. Bowers, J., Button, G., Sharrock, W.: Workflow From Within and Without: Technology and Cooperative Work on the Print Industry Shopfloor. In: The 4th European Conference on Computer-Supported Cooperative Work (ECSCW 95), Stockholm. pp. 51–66. Kluwer Academic Publishers, Dordrecht (1995)
14. BPi. Activity Manager: Standard Program - Standard Forms (Version 1.2). Workflow Management Solutions, Oosterbeek, (2002)
15. Bussler, C., Jablonski, S.: Policy Resolution for Workflow Management Systems. In: Proceedings of the 28th Hawaii International Conference on System Sciences, pp. 831. IEEE Computer Society (1995)
16. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. In: Proceedings of ER '96, pages 438–455, Cottubus, Germany (1996)
17. van Dongen, B., Alves de Medeiros, A.K., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) Application and Theory of Petri Nets 2005. Lecture Notes in Computer Science, pp. 444–454. Springer, Berlin (2005)
18. Du, W., Shan, M.C.: Enterprise workflow resource management. In: 9th International workshop on research issues on data engineering: information technology for virtual enterprises (RIDE-VE'99), pp. 108–115 IEEE Computer Society Press, Sydney (1999)
19. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-aware information systems. Wiley (2005)
20. Ellis, C.A., Keddara, K.: A workflow change is a workflow. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business process management: models, techniques, and empirical studies, vol. 1806, Lecture Notes in Computer Science, pp. 201–217. Springer, Berlin (2000)

21. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Security **4**(3), 224–274 (2001)
22. FileNET. FileNet Business Process Manager 3.0. FileNET Corporation, Costa Mesa (2004)
23. Fischer, L.: (ed.) Workflow Handbook 2003, Workflow Management Coalition. Future Strategies, Lighthouse Point, (2003)
24. Georgakopoulos, D., Hornick, M., Sheth, A.: An overview of workflow management: from process modeling to workflow automation infrastructure. Distrib. Parallel Databases **3**, 119–153 (1995)
25. Grinter, R.E.: Workflow systems: occasions for success and failure. Comput Support. Coop. Work **9**(2), 189–214 (2000)
26. Herrmann, T., Hoffmann, M., Loser, K.U., Moysich, K.: Semistructured models are surprisingly useful for user-centered design. In: De Michelis, G., Giboin, A., Karsenty, L., Dieng, R. (eds.) Designing Cooperative Systems (Coop 2000), IOS Press, Amsterdam, pp. 159–174 (2000)
27. Huang, Y.N., Shan M.C.: Policies in a resource manager of workflow systems: modeling, enforcement and management. Technical Report HP Tech. Report, HPL-98-156, Palo Alto (1999) Accessed at http://www.hpl.hp.com/techreports/98/HPL-98-156.pdf.
28. Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, London (1996)
29. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, vol. 1, EATCS monographs on Theoretical Computer Science. Springer, Berlin (1997)
30. Jensen, K., Rozenberg, G.: (eds.) High-level Petri Nets: Theory and Application. Springer, Berlin (1991)
31. Klein, M., Dellarocas, C., Bernstein, A.: (eds.) Adaptive Workflow Systems, vol. 9, Special issue of the journal of Computer Supported Cooperative Work (2000)
32. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. Int. J. Softw. Tools Technol. Transf. **2**(2), 98–132 (1998)
33. Kumar, A., van der Aalst, W.M.P., Verbeek, H.M.W.: Dynamic work distribution in workflow management systems: how to balance quality and performance? J. Manage. Inf. Syst. **18**(3), 157–193 (2002)
34. Lerner, B.S., Ninan, A.G., Osterweil, L.J., Podorozhny, R.M.: Modeling and managing resource utilization in process, workflow, and activity coordination. Technical Report UM-CS-2000-058, Department of Computer Science, University of Massachusetts (2000) Accessed at http://laser.cs.umass.edu/publications/?category=PROC on 20 March 2005
35. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice-Hall PTR, Upper Saddle River (1999)
36. London Bridge Group. Vectus Application Developer's Guide. London Bridge Group, Wellesbourne, (2001)
37. London Bridge Group. Vectus Technical Architecture. London Bridge Group, Wellesbourne, (2001)
38. Zur Muehlen, M.: Evaluation of workflow management systems using Meta Models. In: Proceedings of the 32nd Hawaii International Conference on System Sciences - HICSS'99, pp. 1–11 (1999)
39. zur Muehlen, M.: Organizational management in workflow applications issues and perspectives. Inf. Technol. Manage. **5**(3–4), 271–291 (2004)
40. Zur Muehlen, M.: Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems. Logos, Berlin (2004)
41. Pallas Athena: Flower User Manual. Pallas Athena BV, Apeldoorn (2002)
42. Reichert, M., Dadam, P.: ADEPTflex: supporting dynamic changes of workflow without loosing control. J. Intelligent Inf. Syst. **10**(2), 93–129 (1998)
43. Reisig, W., Rozenberg, G.: (eds.) vol. 1491, Lectures on Petri Nets I: Basic Models. Lecture Notes in Computer Science. Springer, Berlin (1998)
44. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems: a survey. Data Knowl. Eng. **50**(1), 9–34 (2004)
45. Rosemann, M., Zur Muehlen, M.: Evaluation of workflow management systems — a meta model approach. Australian J. Inf. Syst. **6**(1), 103–116 (1998)
46. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: identification, representation and tool support. In Pastor, O., Falcao e Cunha, J. (eds.) Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05). Lecture Notes in Computer Science, vol. 3520, pp. 216–232, Springer, Berlin (2005)
47. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane (2004)
48. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, (2004)
49. Software-Ley. COSA Activity Manager. Software-Ley GmbH, Pullheim (2002)
50. Staffware. Staffware Case Handler – White Paper. Staffware PLC, Berkshire (2000)
51. Staffware. Using the Staffware Process Client. Staffware, plc, Berkshire (2002)
52. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: Sprague, R. (ed.) Proceedings of the 34th Annual Hawaii International Conference on System Science (HICSS-34). IEEE Computer Society Press, Los Alamitos (2001)