Modeling the Case Handling Principles with Colored Petri Nets

Christian W. Günther and Wil M.P. van der Aalst

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{c.w.gunther, w.m.p.v.d.aalst}@tm.tue.nl

Abstract. *Case handling* is a new paradigm for supporting flexible and knowledge intensive business processes. It is strongly based on data as the typical product of these processes. Unlike workflow management, which uses predefined process control structures to determine what *should* be done during a workflow process, case handling focuses on what *can* be done to achieve a business goal. While classical Petri nets are a good theoretical foundation for workflow management, the data-intensive nature of case handling does not allow for the abstraction of data. Therefore, we use *Colored Petri Nets* (CPNs) as a foundation for case handling. This paper models the key principles of case handling in terms of CPNs and uses state-space analysis and simulation to validate the concepts. Moreover, we also link the CPN model to *process mining* and show that it is possible to rediscover case handling processes based on the event log of a CPN simulation.

1 Introduction

Although workflow management concepts and technology [3, 13, 20, 21] have been applied in many enterprise information systems in the last decade, there appears to be a severe gap between the promise of workflow technology and what systems really offer. As indicated by many authors, workflow management systems (WFMSs) are too restrictive and have problems dealing with change [2, 5, 7, 9, 11, 12, 17, 18, 24].

Most of the workflow management systems consider workflows to be production processes, directly driven by structured process models. Such an approach is solely on the concept of routing, i.e., shifting work among resources based on causal relationship between activities. In [6] it is argued that for some applications the approach is suitable, but that for many other applications the use of control-flow as the primary enactment mechanism may result in the following four problems:

– Distributed handling of work requires its being partitioned, or *straightjacketed into activities*. This fundamental principle, vital for the WFMS, can hardly ever comply with the way workers organize their tasks. Usually activities are performed at a far more fine-grained level than proposed by a process model.

– Typical WFMSs make no distinction between *authorization* and *distribution* of work, i.e., a worker is always offered to pick up *any task he is authorized to do*. This property leads to e.g. overcrowded in-trays for employees in higher positions, as their role typically includes many others.
– Strong control-flow orientation of WFM systems tends to blind out the context of tasks to be performed, most notably data created at earlier points in the process. This leads to the phenomenon of *context tunneling*, i.e., a worker has only access to data deliberately provided, a handicap that hinders efficiency and quality of work.
– The *push-oriented nature of routing* leaves hardly any decision to the user, so that he does not even have a means of making small ad-hoc adjustments to the process. That way workflows become unnecessarily inflexible, and small errors can spawn great problems.

To overcome this problem, we proposed *case handling* as a new paradigm for supporting knowledge-intensive business processes [1, 6]. This paradigm has proven its value in the tool FLOW*er* [1, 8, 22]. This tool is one of the most successful products on the Dutch workflow market and has demonstrated its value in situations requiring more flexibility. The core features of case handling are:

– avoid context tunneling by providing all information available (i.e., present the case as a whole rather than showing just bits and pieces),
– decide which activities are enabled on the basis of the information available rather than the activities already executed,
– separate work distribution from authorization and allow for additional types of roles, not just the execute role,
– allow workers to view and add/modify data before or after the corresponding activities have been executed (e.g., information can be registered the moment it becomes available).

Since case handling is a combination of mechanisms, it is not easy to define this new paradigm in a rigorous manner. The existing definitions given in [1, 6] are too informal to allow for any form of analysis. The purpose of this paper is to "formalize" the case handling concept in terms of *Colored Petri Nets* (CPNs) [15, 19]. CPNs are a natural extension of the classical Petri net [23]. There are several reasons for selecting CPNs as the language for modeling the case handling paradigm. First of all, CPNs have formal semantics and allow for different types of analysis, e.g., state-space analysis and invariants [16]. 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[1] – a graphical environment to model, enact and analyze CPNs. The model was created by one person, having a software engineering background and basic knowledge about classical Petri nets, in about four man-weeks. A significant amount of this time was spent

[1] CPN Tools can be downloaded from wiki.daimi.au.dk/cpntools/.

getting acquainted with CPN Tools as an application in general, and Standard ML for e.g. functions in particular.

The remainder of this paper is organized as follows. First, the case handling paradigm is introduced, emphasizing its specific features in contrast to traditional Workflow Management. Subsequently, Section 3 introduces a CPN model representing the basic case handling functionality in an abstract manner, including the results of a state space analysis showing model correctness. Section 4 discusses the applicability of the presented model by inspecting its alignment to an industrial case handling system and discussing limitations. Further, an extension to the model is presented that allows for generating artificial enactment logs for process mining research, followed by a conclusion.

2 Case Handling

In contrast to the strongly process oriented view of production workflow, emphasizing the routing between atomic activities, the case handling paradigm focuses mainly on the *case* itself. The case is the primary object to be manufactured in any kind, e.g. the outcome of a lawsuit or the response to a customer request. Resulting from that, single activities diminish in importance in favor of the larger context. They are no longer considered atomic steps that have to be performed in an "all or nothing" manner, but rather serve as logical partitions of work between which a transition from one worker to another is possible.

As in traditional WFM there exists a set of precedence relations between single activities making up a process, using well-known patterns [4] for that. However the primary driver for progress is no longer the event of explicitly finishing activities but the availability of values for data objects. While production workflow clearly separates the process from associated data, case handling integrates both far more closely, using produced data not only for routing decisions but also for determining which parts of the process have already been accomplished. With case handling, each task has associated with it data objects for three distinct purposes, while the first association is between a task and all data objects that are accessible while performing it. Further on, all data objects that are *mandatory* for a task have to be set (i.e., bound to a value) before the task itself is considered to be accomplished by the system. Finally, every data object can have a random number of tasks to which it is *restricted*, meaning that it can only be altered while performing one of these tasks. The *mandatory* and *restricted* properties are independent from each other, however reason dictates to have the last (in the sense of a causal chain) task featuring a data object as *restricted* also declare it as *mandatory* (to make sure it will be provided). User-interactive tasks are connected to a *form*, each providing access to a selection of data objects. Note that one form can be associated with multiple tasks; on the other hand it is also possible to associate a form to the case itself, so that it can be accessed at any point in time.

To introduce the case handling principles, Figure 1 shows a simplified example of a case type, the case handling analogy to a workflow process definition:

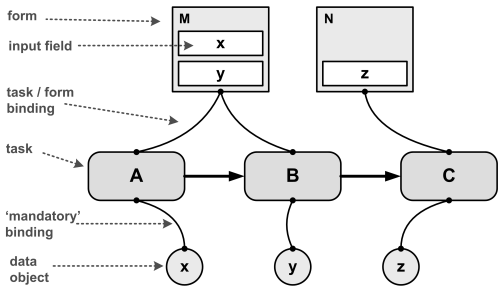

Fig. 1. Simplified example case type

Three tasks A, B and C are making up the process, sequentially chained by causal relationships denoted by connecting arrows. Their *mandatory* relationships to the three data objects x, y and z below are denoted by curved arcs, correspondingly they are associated with the forms M and N above. As can be seen in the illustration, tasks A and B share the same form M, providing access to data objects x and y. If a properly authorized worker now starts handling task A, the associated form M will open and he will start providing values for the presented data objects. In a traditional WFMS activity A would not be finished before form M is closed, however the case handling system regards A as finished as soon as a value for x has been provided (and confirmed appropriately), automatically enabling task B in the background. If the worker would now close form M, another employee could pick up the case where he left it, starting task B, which would provide the same form M with x having a value filled in (that could now be changed again). Another possibility is, however, that the first worker keeps on handling the form, providing also a value for y. This would correspondingly trigger the *auto-completion* of task B (as all associated mandatory data elements, in this case only y, have been provided) and activate task C. Note that if a worker closes a form after filling out only parts of the mandatory data fields of a task, despite the task not being considered finished data already entered is not lost but will be presented to the person continuing work on that task.

Such closely intertwined relationship between data and process obviously abandons their, often unnatural, separation so rigidly pursued in traditional workflow management. With the status of case data objects being the primary determinant of case status, this concept overcomes a great deal of the problems described in the introduction:

- Work can now be organized by those performing them with a far higher degree of freedom. Activities can either be performed only partly, without losing intermediary results, or multiple related activities can be handled in one go, surpassing the considerably weakened border between single tasks.
- The phenomenon of *context tunneling* can be remedied by e.g. providing overview forms directly associated with the case. Every authorized worker

can consult such form at any point in time, ensuring he is aware of the context where necessary.

– Routing is no longer solely determined by the process model. Case types can be designed in such a way, that multiple activities become enabled concurrently, providing different ways of achieving one goal. It is up to the user to decide which way to go, with the system "cleaning up behind", i.e., disabling or auto-completing tasks that have not been chosen.

In addition to the *execute* role, specifying the subset of resources allowed to handle a specific task, the case handling paradigm introduces two further roles crucial for operation. The *skip* role allows workers to bypass a selected task, which could be interpreted as an *exception*. When one thinks of real business processes an exception, like skipping an activity that deals with thoroughly checking the history of a client before granting a mortgage for well-known and trusted clients, is likely to occur quite frequently. The ability to grant the *skip* role to a senior worker renders the necessity for implementing such bypass obsolete, thus greatly simplifying the whole case type. It has to be noted that in order to skip a task all preceding tasks that have not been completed yet have to be skipped (or completed) beforehand. Traditional workflow definitions use loops for repeating parts of the process, e.g. because they have not yielded an expected result. In a case handling system, such construct has been made obsolete as well by the introduction of a *redo* role, enabling its bearer to deliberately roll the case's state back and make a task undone. In doing so, the values provided for data objects during this task are not discarded but merely marked as *unconfirmed*, so that they serve as kind of template when re-executing the affected task. Similar to skipping, before a task can be *redone* all subsequent tasks that have already been completed need to be rolled back as well before. Roles in a case handling system are *case specific*, i.e., having assigned the role "manager" for a case type A does not imply that one can play the same role for another case B. They can be specified in form of a *role graph*, where single role nodes are connected to each other by arcs, symbolizing *is-a* relationships; i.e., being authorized to play a role also implies the authorization to play all roles connected as child nodes.

Intertwining authorization with distribution of activities has been one major flaw of traditional workflow technology. In a case handling system, the former *in-tray*, i.e., a list of all activities the user is authorized to perform and that he can choose from, has been replaced by a sophisticated *query mechanism*. This tool can be used to look for a specific case, based on certain features (e.g. case data, or enactment meta-data like the start date of a case instance). Moreover it can be used to create predefined queries tailored to each worker (or, group of workers). A manager is no longer constantly flooded with all possible activities that he can perform, but only those which require a certain role, or e.g. case instances of an order where the combined value exceeds $1000. Obviously the query mechanism can also be used to perfectly imitate a classic in-tray, be it required.

3 CPN Model

This section introduces the CPN model of a case handling system, thus making the case handling paradigm explicit. We first discuss the color sets and then show the top level view of the model. The top level model contains two substitution transitions, which are also discussed. Finally, the model is evaluated using the state space tool of CPN Tools.

3.1 Color Sets

Before showing the top level view of the model, we present some of the color sets used in the model (cf. Figure 2). Data elements are represented by the color set `DATUM` which is composed of a string that denotes its (unique) name, a boolean flag symbolizing if it is enabled (i.e., if its value can currently be set or unset), and a second boolean flag representing whether the value has actually been set. The place `all_data` contains complete information about all data elements as one single token of color set `DATUMLIST`, a list of `DATUM` instances, serving as central repository and interface between all three parts of the model. Color set `RESOURCE` is composed of a string containing the unique name of the resource, and a list of strings denoting the roles this resource can play. The most complex color set within this model is the `TASK`, composed as follows: One string contains the unique name of the task, followed by a list of strings representing the (names of the) data elements that can be accessed and another list of strings representing the mandatory data elements of this task (i.e., those that have to be set in order to complete a task). Two further string lists contain the names of preceding and successive tasks, thus making up the control flow of the case, and three strings specify the respective roles necessary for executing, skipping and redoing the task.

```
color STRINGLIST = list STRING;

color DATUM = product
  (* name *) STRING *
  (* enabled *) BOOL *
  (* isset *) BOOL;
color DATUMLIST = list DATUM;

color ROLE = STRING;

color RESOURCE =  product
  (* name *) STRING *
  (* roles *) STRINGLIST;

color MUTEX = unit with null;

color TASKSTATE =
  with initial | enabled | finished;
```

```
color TASK = product
  (* name *) STRING *
  (* data *) STRINGLIST *
  (* mandatory *) STRINGLIST *
  (* previous *) STRINGLIST *
  (* successors *) STRINGLIST *
  (* exec role *) STRING *
  (* skip role *) STRING *
  (* redo role *) STRING;

color TASKxRESOURCE =
  product TASK * RESOURCE;

color TASKENTRY = product
  (* name *) STRING *
  (* state *) TASKSTATE;

color TASKLIST = list TASKENTRY;
```

Fig. 2. Color set definition used in the CPN model

3.2 Top-Level View

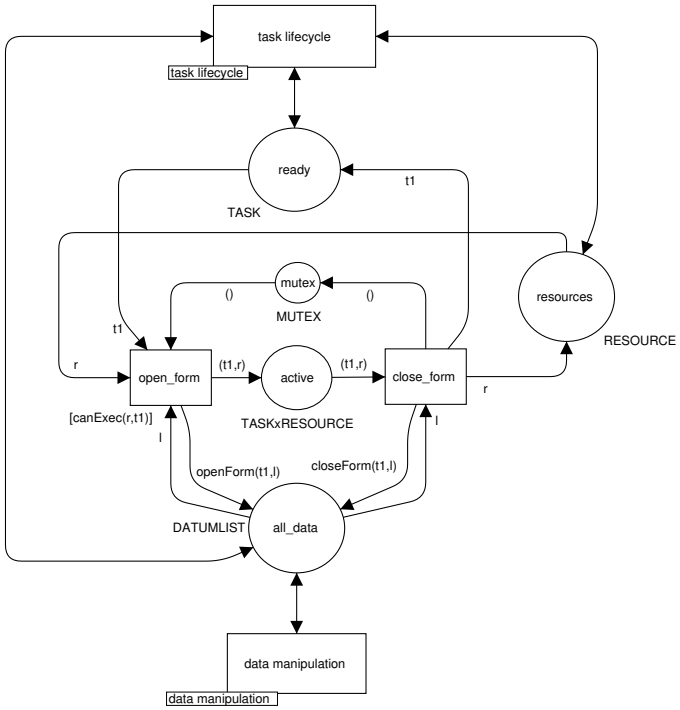

Fig. 3. Top-level view of the model

All basic functionality available to the user of a case handling system is included within the top level view. The `ready` place is the main interface to one further part of the model describing the lifecycle of tasks; when a task is ready to be handled it is transported to this place by the system, respectively tasks whose postcondition has been satisfied get removed from this place by the system. Users are now able to activate tasks that are being located in place `ready` for handling, which usually means opening the respective form in order to change associated data. This activity corresponds to firing transition `open_form`, consuming a task from `ready` and a resource token from the central repository place, both being stored intermediately in place `active` (as a product of both colors). When fired, `open_form` will also consume the token from `all_data` and set all data elements defined in the opened task as enabled. Subsequently firing transition `close_form` will adversely disable all data elements specified by the now closed task, transport the task token itself back into `ready` and free the associated resource, i.e., put it back into the central resource repository. Note that opening and closing forms is the only way to enable or disable data elements

for manipulation and, the other way round, apart from that no change in the overall state is performed.

One thing that has to be noted is that case handling systems usually not allow for real concurrency. To avoid context tunneling it will provide access to all case data, and therefore limit parallel updates of data values. If several users could open forms on one single case in parallel, the outcome of this would be impossible to determine, as they could enter contradictory values for one data field on different forms. This behavior is modeled by place `mutex` which contains exactly one black token. Once this token has been consumed by firing `open_form`, no further form can be opened unless firing `close_form` has produced a new token in `mutex`.

To avoid context tunneling and provide to the experienced user a higher degree of freedom, case handling systems allow to define forms that are associated directly to the case (in contrast to task-associated forms). These forms can be opened at any time and allow direct access to a random choice of case data elements. In this model, case forms are represented by tasks having no predecessor and successor tasks and no mandatory data elements. They reside in the `ready` place and will not leave it, due to their not being connected to other tasks from a control-flow point of view.

3.3 Data Manipulation

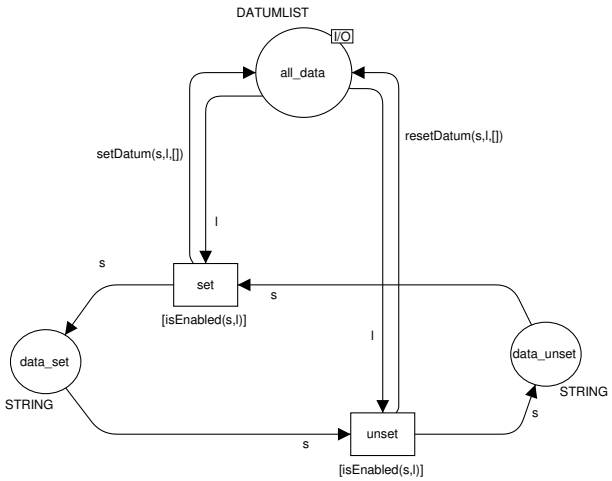

Fig. 4. Data manipulation subsection of the case handling model

This part of the model represents the setting and unsetting of data values dependent on their being enabled by currently open forms. Single data elements are represented by simple string tokens (containing their name) and can either

reside in places `data_set` or `data_unset`, so that their state is symbolized by their position within the model. A transition between the states of being set or unset — i.e., between the two respective places — can be achieved by firing transitions `set` and `unset`. The precondition for firing these transitions is, that for the data element to be (un-)set the "enabled" flag is set to true within place `all_data`, a condition checked by function `isEnabled`. In effect, firing either `set` (`unset`) executes function `setDatum` (`resetDatum`), toggling the boolean flag denoting the value-bearing status of the respective data element within `all_data`.

3.4 Task Lifecycle

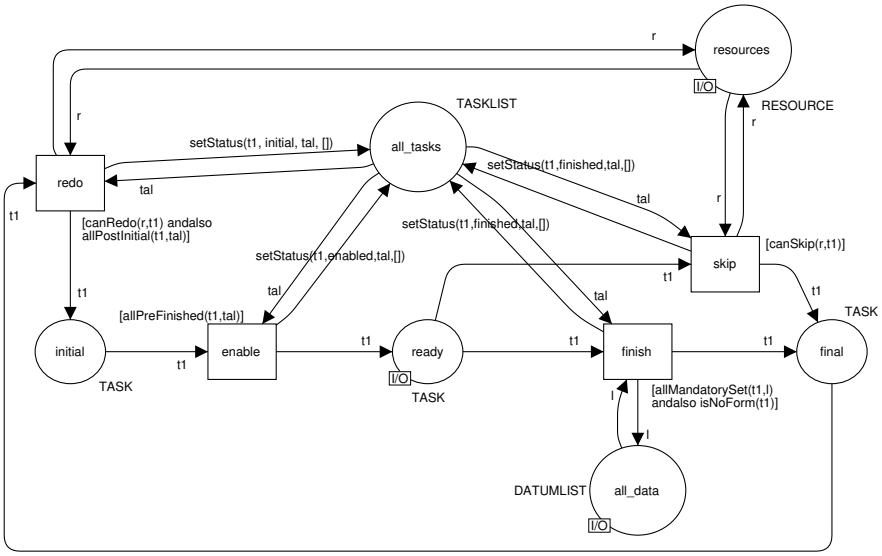

Fig. 5. Part of the model representing the task lifecycle

Handling the lifecycle of tasks is essential to the case handling mechanism, for this is possibly the point where such system differs most substantially from a conventional WFMS. The model is built to reflect the current lifecycle status of each task as its respective position in one of three places of type `TASK` (or four, if you count the `active` place in the top view part). Any task is either in state (place) `initial`, `ready` or `final`. One place `all_tasks` serves as a central repository for task state information, replicating this for the sake of model readability[2]. State transitions for tasks are dependent on the states of

[2] It would have been possible to test multiple places as precondition for firing a transition instead of just polling `all_tasks`. However, by using this central replication of state information it is possible to avoid a large amount of arcs crossing each other.

their respective predecessor and successor tasks, as well as it is dependent on the state of the case's data elements, i.e., which of them have already been set. When a case is started, all tasks contained will reside within place `initial`, waiting to be enabled by firing transition `enable`. The precondition for that is that all tasks that are direct predecessors have already been finished, it is checked by function `allPreFinished` which therefore retrieves the central token from `all_tasks` and puts it back, not before having changed the respective status of the enabled task.

After having been enabled the task resides at the `ready` place, from which basically three possibilities exist for progressing further. The task token can transcend into the top-level part of the model by being activated, i.e., opening its adjacent form and starting to change data values associated. This will, however, finally result in the form being closed again, returning the task into place `ready`. If the user decides so and has sufficient rights (i.e., a resource having the necessary role is available) he can deliberately `skip` a task (implemented by a transition of that very name), transporting the respective token directly into place `final`. This will adjust the task's status within the central repository `all_tasks` and free the resource again immediately afterwards.

The usual way, however, for a task to progress from `ready` to `final` is by firing transition `finish`. This transition gets enabled as soon as all mandatory data elements of a task have been set, further it is ensured that the respective task is "real", and not in fact a case form (by making sure it has predecessor and successor tasks) and the task state is adjusted in `all_tasks`. By solely depending on the status of data elements for finishing tasks this transition also implements the "autoskip" functionality of case handling systems, i.e., the possibility to finish tasks without even having touched them, solely by satisfying their mandatory data requirements otherwise (e.g. using case forms).

Redoing tasks requires, opposite to enabling them, that all successors of the respective task are in place `initial` (i.e., that they have previously been rolled back). The `redo` transition checks this property, and further needs to acquire a suitable resource that is freed immediately afterwards. The possibility to redo tasks closes the circular lifecycle, allowing tasks to progress through this an arbitrary number of times, under the sole premise that the basic (half-)ordering of tasks be maintained.

3.5 Evaluation

In order to both verify the correctness of the model and ensure alignment with the case handling principles that were intended to be modeled, a state space analysis of the model was performed within CPN Tools. However, before such analysis was feasible we were forced to reduce the size of the example case handling process and abstract from forms. Without these changes, CPN Tools was unable to construct the full state space. A token from place `ready` (symbolizing a form directly associated with the case) was removed and the represented process was simplified into a case handling process of only two consecutive steps `task1` and `task2`, i.e., the initial state of place `initial` was changed and the token in

place `all_tasks` was adjusted accordingly as well. However, the basic principles of case handling remain untouched by these alterations. As a result, the analysis in remainder remains relevant for other case handling processes.

Place	Upper Integer Bound	Lower Integer Bound
active	1	0
all_data	1	1
mutex	1	0
resources	3	2
ready	1	0
data_set	4	0
data_unset	4	0
all_tasks	1	1
final	3	1
initial	3	1

Table 1. Integer bounds of simplified case handling model

Boundedness Properties — One property the state space analysis can yield is the upper and lower integer bound of tokens per place, i.e., the maximum and minimum number of tokens of the appropriate color each place can contain for a given initial marking [14]. With respect to the scenario described above, the results are documented in Table 1. One first property that can be noted is that every place has an upper bound, i.e. the net is bounded. This corresponds to our expectations, as the model is not intended to generate additional tokens, so that the overall number of tokens contained within the complete net is expected to remain unchanged by firing any transitions[3]. Further it can be observed that places `all_tasks` and `all_data` will always contain exactly one token, as these are to serve as central information repositories which are not to be moved permanently. The difference in upper and lower bounds for place `resources` of merely 1 corresponds to the upper bound of 1 for place `active`, emphasizing the nature of a case handling system locking the complete case for exclusive access to one user at a time. Finally, the lower bounds of 1 for places `initial` and `final` are due to the virtual `START` and `END` tokens of color `TASK` which remain in their respective places infinitely as guaranteed process boundaries.

Home Properties — The case handling paradigm allows for a maximal degree of freedom with respect to navigating within a process. Given that he has sufficient permissions, the user can freely move back and forth within a case type's

[3] An exception to this is place `mutex`, which is empty as long as an activity or form is open.

process model by executing, skipping and redoing tasks at will. As extreme examples consider at the one hand finishing a case by merely skipping all available tasks, and on the other hand rolling an otherwise completed case completely back by redoing the tasks contained in reverse order. This behavior, i.e., the ability to reach any state from every other possible state, is reflected in the state space analysis labeling all possible markings of the net as Home Markings, i.e., markings that it is always feasible to reach [14].

Liveness Properties — Liveness denotes the remaining active of a set of binding elements, i.e., every transition can become enabled by firing an arbitrary number of transitions[14]. In contrast to this, a dead marking denotes a state of the net in which no transition is enabled. With respect to the model presented, the only acceptable dead marking, i.e., a state in which no further action is possible, would be a state in which the process is completely finished. However, given the fact noted above that case handling provides unlimited navigation within the process by means of skipping and redoing tasks, neither should there exist dead markings nor dead transitions (i.e., single transitions that can become permanently disabled by any firing sequence) for the model. This expectation has been acknowledged by the state space analysis, yielding no dead markings and denoting all transition instances as live.

Fairness Properties — Fairness is an indicator of how often different binding elements occur[14]. For the given model, the state space analysis denotes transitions `enable` and `redo` as fair, so an infinite number of enablings for these transitions implies an infinite number of occurrences. This corresponds to the intended behavior, as these transitions will actually remain enabled until fired. With the exception of `close_form`, all further transitions are just, i.e., persistent enabling of these implies an eventual occurrence. This property can be deduced from the free-choice character of the model, i.e., choices between two concurrently enabled transitions depend solely on user decision and are also intended to reflect these. Finally, transition `close_form` has no fairness. This is due to the fact that, in case this transition is enabled, it is always possible to fire `set` and `unset` in an infinite, alternating sequence, so that firing `close_form` can be effectively evaded. However, such behavior would correspond to a user having one specific form open infinitely long and keeping on changing values without eventually closing it again. Therefore this property reflects an intended model behavior, and a real-life occurrence of such potential infinite loop can safely be ruled out by common sense.

4 Applicability

The purpose of this section is to probe the real-life applicability and relevance of the presented model. In order to more illustrate the concept of case handling, and also to bridge the gap between the abstract model presented and real life

application, the first section compares the CPN model to the real case handling system FLOW*er* [1, 8, 22]. The second part discusses limitations of the presented model and discrepancies with respect to FLOW*er*, while the last part introduces an extension of the model for creating enactment logs during simulation.

4.1 Mapping to FLOW*er*

In order to more illustrate the concept of case handling, and also to bridge the gap between the abstract model presented and real life application, this section compares the CPN model to the real case handling system FLOW*er*.

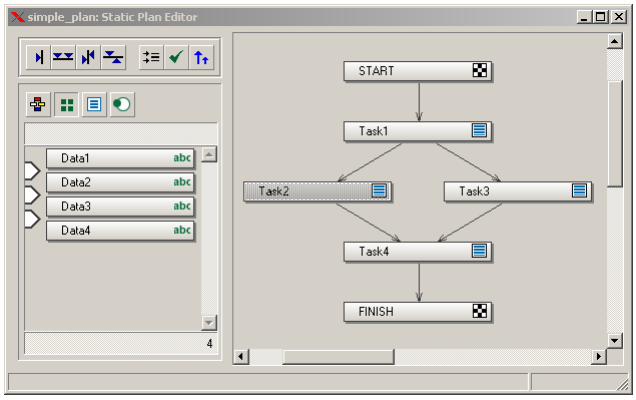

Fig. 6. Process model ("plan") in FLOW*er*

FLOW*er* features a graphical process designer tool (FLOW*er* Studio), in which so-called `case types` can be defined, Figure 6 shows a screenshot of this application with a simple example process[4]. On the left side four rectangular boxes can be seen, symbolizing the four data objects *Data1* to *Data4* used in this case type. To the right, the process structure is depicted, showing the tasks *Task1* to *Task4* as rectangular nodes, connected with arcs denoting causal relations between them. The start and end of the process is marked with so-called *Milestones*, which are no real tasks but symbolize defined states of the process and have been included here to make a connection to the fixed `START` and `END` tokens within the CPN model.

Apart from the relationships visible in Figure 6, the following has been defined: All four tasks use the same form (*Form1*), providing access to all four data objects; this form is also connected to the case type itself, i.e., it can be accessed at any time during handling a case. *Task1* has *data1* as mandatory, *Task2* and *Task3* both have *data2* and *data3* specified as mandatory and *Task4* requires *data4* respectively.

[4] In FLOW*er*, the process part of a case type is called a *plan*.

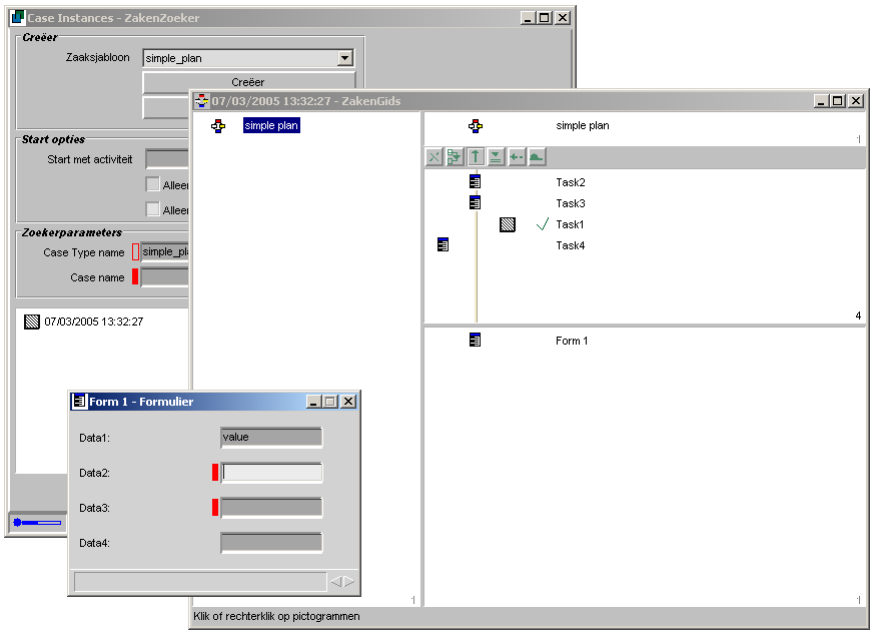

Fig. 7. User perspective of the FLOW*er* system

Figure 7 shows the user interface of FLOW*er*. In the front, the single form has been opened in reaction to executing *Task1* and a value has been provided and acknowledged for *data1*. The window in second order shows the main user interface of FLOW*er* which provides an overview of the case currently handled by the user. On the left, the single plan, i.e., process model, of the currently handled plan has been selected, thus it is also displayed on the top third of the right part. In the middle part of the right side the so-called *wavefront* is displayed, which is mainly a vertical line. Tasks that are being displayed on the left of this line are not yet ready to be executed, enabled tasks are right on the wavefront and completed tasks are shown to the right of it. Thus, tasks "travel" from left to right over the wavefront correspondingly to their lifecycle status (and they move back to the left again when they are being redone). Below the wavefront part of the interface, *Form1* is explicitly depicted and can be opened any time while handling this case.

As in this example a value has already been provided for *data1*, the first task (*Task1*) has already been finished (as *data1* was the only mandatory data object for this task) and has moved to the right of the wavefront. Corresponding to this, *Task2* and *Task3* have been enabled and are positioned right on the wavefront. A closer look at the opened form reveals red square icons to the left of the entry fields for *data2* and *data3*: Both *Task2* and *Task3* that are now enabled have set these data objects as mandatory, thus the system now signals (by the square

indicators left to each input field) that, by providing values for these fields, the next step could be accomplished.

Notice the non-intrusive nature of guiding user interaction performed by the case handling system, which is not enforcing or suggesting any specific behavior but rather presenting possible options. The user is not controlled by the system in a push-oriented manner, but he potentially has a great degree of freedom in his actions (depending on the design of the case type). Meanwhile the system will constantly observe which parts of the process have already been accomplished and track progress, providing more of a help in direction than restricting to predefined paths.

Table 2. Model place contents in discussed state

Place	Tokens
all_data	`1‘[("data1", true, true), ("data2", true, false), ("data3", true, false), ("data4", true, false)]`
all_tasks	`1‘[("START", finished), ("task1", finished), ("task2", enabled), ("task3", enabled), ("task4", initial), ("END", initial)]`
initial	`1‘("END", [], [], ["task4"], [], "", "", "")++ 1‘("task4", ["data1", "data2", "data3", "data4"], ["data2", "data3", "data4"], ["task2", "task3"], ["END"], "role3", "role3", "role2")`
ready	`1‘("task3", ["data1", "data2", "data3", "data4"], ["data2", "data3"], ["task1"], ["task4"], "role1", "role1", "role2")`
active	`1‘(("task2", ["data1", "data2", "data3", "data4"], ["data2", "data3"], ["task1"], ["task4"], "role2", "role2", "role3"), ("resource1",["role1","role2"]))`
final	`1‘("task1", ["data1", "data2", "data3", "data4"], ["data1"], ["START"], ["task2", "task3"], "role1", "role1", "role2") ++ 1‘("START", [], [], [], ["task1"], "", "", "")`
resources	`1‘("resource2",["role2","role3"]) ++ 1‘("resource3",["role3","role1"])`
data_set	`1‘"data1"`
data_unset	`1‘"data2" ++ 1‘"data3" ++ 1‘"data4"`

If the example case type and situation as described above is translated into the introduced CPN model, the marking of the net after executing *Task1* and providing a value for *data1* is given in Table 2. In the token for `all_data` it can be observed that all four data elements have been enabled (with their second element, a boolean, set to `true`, while merely `data1` has its third element set to `true`, corresponding to its having been set. This last information is also explicitly reflected in the distribution of tokens between `data_set` and `data_unset`.

4.2 Limitations

The mapping performed in the last section shows that the presented CPN model is suitable for introducing and analyzing the basic features of the case handling paradigm, enabling the implementation and analytic enactment of simple case types. However, compared to the presented commercial system FLOW*er* it has several subtle discrepancies and limitations, which shall be discussed within this section.

Regarding case type design, the model does not allow for alternative branches within the process structure. Such feature would require the interpretation of Boolean statements on data objects, which is beyond the scope of the presented model. Another fundamental limitation is concerning process structure as well, for constructs like arbitrarily instantiated sub-processes[5] cannot be represented with this model. This does, however, have no influence on the correctness of the model, for these can be interpreted as abstract tasks executed in parallel.

Naturally, FLOW*er* as a commercial system allows for a much wider range of task definitions beside form actions. Tasks can also use database interactions, arbitrary code execution and other non-interactive methods of data processing. On the conceptual level of the model these differences are, however, completely irrelevant — all task types available in FLOW*er* share the same basic property relevant for the scope of the model, i.e., they all access and potentially modify case data objects.

Other constructs present in FLOW*er* can be seen as helpful abstractions, which can be implemented using the model by breaking them down into more low-level constructs. One example are role hierarchies; these are not supported by the model, but can be implemented by granting every resource that has role r also each child role of r. Regarding the *restricted* feature of data objects, which is also not directly implemented in the model, it is possible to make the respective data object available only for those tasks to which it shall be restricted, resulting in the very effect.

One aspect in which the behavior of the presented model deviates significantly from that of FLOW*er* is concerning task lifecycle transition. While the model treats task and associated form as unity, this association is implemented much less coherent in FLOW*er*. While, after providing a value for *data1* in the example of the last section, *Task1* has already transitioned to the final state, i.e., to the right of the wavefront, in the model this transition can only happen after the form has been closed. This discrepancy extends onto the time of enabling subsequent *Task2* and *Task3*, which is self-evident as they have to wait until *Task1* is finished. While this difference shows in the delayed transition of tasks, it has no influence on the general order of these transitions, such that the overall behavior of the model is consistent with FLOW*er*. Due to this property, results gained from the model can be applied with little to no restriction.

[5] These serve for executing a specific subprocess multiple times in parallel, whereas the number of subprocess instances is only determined during super-process enactment (e.g., collecting witness statements for an accident).

4.3 Process Log Synthesis

The presented CPN model obviously does not directly represent a business process, but it rather describes the abstract process of handling, or executing, such business processes within a case handling system. The executed process is thus not explicitly visible in this *meta* model, it is encoded in tokens of type *TASK*. Only when the model is executed, the handled process becomes observable.

To make such observation easier to achieve, and to allow for a more detailed analysis of the handled processes, the model was enhanced with the logging extensions of CPN Tools[10]. These are three ML functions, `createCaseFile`, `addATE` and `calculateTimeStamp`, that create process enactment logs of a model in MXML, the ProM[6] format, when the model is simulated in CPN Tools. An extra transition has been added to the model, activating the logging function `createCaseFile(int caseId)` before each execution of the model. This will create a new log file containing all events recorded for this particular simulation run (i.e., case), while the parameter *caseId* has to be manually set to a unique number in order to distinguish between different cases.

The actual log events, describing the life-cycle change of the task in question, are recorded by the function `addATE(int caseId, String transitionName, ListOfStrings eventyType, StringTimestamp timestamp, String originator, ListOfStrings data)`, which is called when firing transitions *enable*, *finish*, *skip*, and *redo*. Parameter *caseId* is set according to the value provided in `createCaseFile`, while the value for *transitionName* corresponds to the task name contained in the processed *TASK* token. The *eventType* is determined by the fired transition, i.e. transition *enable* will set the event type to *schedule*, *finish* to *complete* and so on. Notice that while the names are different, this is only due to a different terminology between the case handling principle and the MXML format, the semantics are perfectly in line. For setting the *timestamp* parameter, the helper function `calculateTimestamp` is called, creating ascending, logical timestamps. Finally, the *originator* parameter is set to the name of the *RESOURCE* token used (if any, otherwise it is left blank), while the *data* field is left blank.

Multiple simulations of the same process model can be aggregated into one log file, using a plugin in the ProM*import* framework[6]. This is an application that allows for importing MXML logs from all sorts of process-aware information systems. The aggregated log is then ready to be analyzed using the wide palette of Process Mining techniques available within ProM. Figure 8 shows the results of mining 20 simulation runs of the model with the Alpha algorithm. The process model encoded in the *TASK* tokens corresponds to the case type shown in Figure 6, except for one minor adaption: In order to ease mining and produce a sound WF-Net a task *INIT* has been prepended to the process. It cannot be skipped and rolled back, thus ensuring exactly one defined start condition without incoming arcs.

[6] ProM is the process mining framework. Both ProM and ProM*import* can be freely downloaded at http://www.processmining.org/.

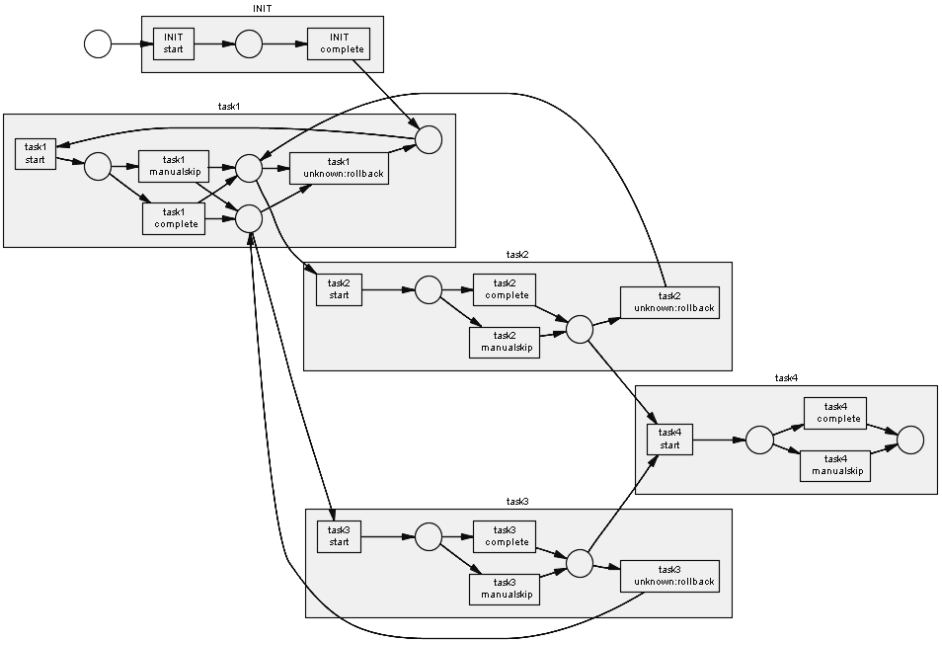

Fig. 8. Example process model mined from simulation logs

It is worthwhile pointing out that the mined process model, which is a WF-Net satisfying the soundness criteria, describes possible paths of execution in a very concise and accurate manner. For each task a similar sub-process has been discovered, showing the typical case handling lifecycle of a task with scheduling, skipping or completing, and rolling back. This accuracy allows for the model to be used for the sake of synthetic case handling log creation by simulation. In order to research new methods for Process Mining in the field of case handling systems a wide variety of enactment logs is required, and log synthesis by simulation can deliver these in a fast and automatized manner. Notice further that the ProM*import* framework also includes a plugin for importing logs from FLOW*er*. Actual case handling processes can be remodelled in the CPN model to research all potential enactment paths in synthesized logs, and the results can be compared to those obtained from analyzing real FLOW*er* logs.

5 Conclusion

In this paper we have discussed the deficiencies contemporary Workflow Management Systems suffer from and that hinder their wider application in industry, boiled down to four crucial problems. Subsequently, the case handling paradigm has been presented as potential remedy to these structural problems. Using the technique of Colored Petri Nets, an abstract model of case handling has been

introduced, incorporating all significant features of the way a case is being processed in such system.

This model can on the one hand be employed to understand the functionality of case handling and trace process enactment down to fine-grained steps. On the other hand, close alignment of the model to a real case handling system, FLOW*er*, has been shown in all significant aspects. Together with a state space analysis, proving basic correctness and expected behavior of the model, these properties can be extended onto the principles of case handling as a whole. Taking into account the mentioned limitations of the model and its discrepancies with respect to a real system like FLOW*er*, it can be employed to conduct more thorough research on specific properties of case handling. Especially the extension of the model with logging capabilities can provide valuable artificial logs, which are important for process mining research on case handling systems.

6 Acknowledgements

This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

The authors would like to thank Kurt Jensen for his support in the early stages of this article and his comments, and Ana Karla Alves de Medeiros for implementing the logging extensions to CPN Tools. Last but not least, the authors would like to thank the anonymous reviewers for their useful remarks.

References

1. W.M.P. van der Aalst and P.J.S. Berens. Beyond Workflow Management: Product-Driven Case Handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001)*, pages 42–51. ACM Press, New York, 2001.
2. W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
5. W.M.P. van der Aalst and S. Jablonski. Dealing with Workflow Change: Identification of Issues and Solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.
6. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
7. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 218–234. Springer-Verlag, Berlin, 2000.

8. Pallas Athena. *Case Handling with FLOWer: Beyond workflow*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.
9. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. In *Proceedings of ER '96*, pages 438–455, Cottubus, Germany, Oct 1996.
10. A.K. Alves de Medeiros and C.W. Günther. Process mining: Using cpn tools to create test logs for mining algorithms. *http://is.tm.tue.nl/research/processmining/tools/ProM/cpnToolConverter.zip*, 2005.
11. C.A. Ellis and K. Keddara. A Workflow Change Is a Workflow. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 201–217. Springer-Verlag, Berlin, 2000.
12. T. Herrmann, M. Hoffmann, K.U. Loser, and K. Moysich. Semistructured models are surprisingly useful for user-centered design. In G. De Michelis, A. Giboin, L. Karsenty, and R. Dieng, editors, *Designing Cooperative Systems (Coop 2000)*, pages 159–174. IOS Press, Amsterdam, 2000.
13. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
14. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992.
15. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
16. K. Jensen and G. Rozenberg, editors. *High-level Petri Nets: Theory and Application*. Springer-Verlag, Berlin, 1991.
17. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems*, Seattle, Washington, November 1998.
18. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, volume 9 of *Special issue of the journal of Computer Supported Cooperative Work*, 2000.
19. L.M. Kristensen, S. Christensen, and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, 1998.
20. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
21. D.C. Marinescu. *Internet-Based Workflow Management: Towards a Semantic Web*, volume 40 of *Wiley Series on Parallel and Distributed Computing*. Wiley-Interscience, New York, 2002.
22. Pallas Athena. *Flower User Manual*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.
23. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
24. M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In R. Sprague, editor, *Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-34)*. IEEE Computer Society Press, Los Alamitos, California, 2001.