Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes

F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, H.M.W. Verbeek

Eindhoven University of Technology
P.O. Box 513, 5600MB Eindhoven, The Netherlands
e-mail: {f.gottschalk,w.m.p.v.d.aalst,m.h.jansen-vullers,h.m.w.verbeek}@tue.nl

Received: date / Revised version: date

Abstract. Protos is a popular tool for business process modelling used in more than 1500 organizations. It has a built-in Petri-net-based simulation engine which shows key performance indicators for the modelled processes. Reference process models offered for Protos reduce modelling efforts by providing generic solutions which only need to be adapted to individual requirements. However, the user can neither inspect or interact with simulations running in Protos, nor does Protos provide any explicit support for the adaptation of reference models. Hence, we aim at a more open and configurable simulation solution. To realize this we provide two transformations from Protos models to colored Petri nets (CPNs), which can be executed by CPN Tools. The first transformation enables the usage of the extensive simulation and measuring features of CPN Tools for the simulation of Protos models. The second transformation creates colored Petri nets with dedicated features for process configuration. Such configurable process models can be restricted directly within the process model without changing the model's structure and provide therefore dedicated adaptation features for Protos' reference process models.

1 Introduction

Today "process thinking" has become a mainstream organizational practice [22]. Business process models provide a graphical and systematic view on organizational processes [17]. Various tools for business process modelling have been developed since the late nineties [2]. One popular tool is Protos from the company "Pallas Athena". Currently it is used by about 1500 organizations in more than 20 countries. E.g., more than half of all municipalities within the Netherlands use Protos for the specification of their in-house business processes [26].

Most providers of modelling tools, and, e.g., also all dominant enterprise system vendors provide reference models with or for their software. Reference models are supposed to reduce the modelling efforts by providing generic solutions that just need to be adapted to individual requirements [5,9,10,11,20,21]. Pallas Athena provides several sets of reference process models implemented in Protos. As an example, there is a set of about 60 reference process models for municipalities. These are ordinary Protos models depicting common processes. The municipality or organization buying such a reference model can adapt the models to its individual requirements, avoiding the huge effort of building process models from scratch. However, it is quite important to note that neither Protos nor any other popular process modelling tool provides an explicit support for the adaptation of reference models, i.e. these tools do not provide any constructs or mechanisms that highlight where and how a syntactically and semantically valid change of a given model is possible or which changes are impossible [18].

Figure 1 depicts a reference process model for the handling of objections against parking tickets[1]. If an objection is received within the corresponding deadline it is checked for its admissibility. In case it is not admissible, a supplement is requested. If this is received within the new deadline or if a supplement was not needed, the parking administration is consulted, reasons for approval/refusal as well as a settlement are drawn up, and the judgement is sent to the objecting citizen. Otherwise the objection times out and is refused directly as it is in case the objection was not received within the deadline. That means, although it is possible to specify in Protos if a task has an XOR or an AND joining/splitting semantics, in this example all splits and joins of the process flow are in an XOR relation.

[1] The model is motivated by one of Pallas Athena's reference models, but specially build for the purpose of this paper.

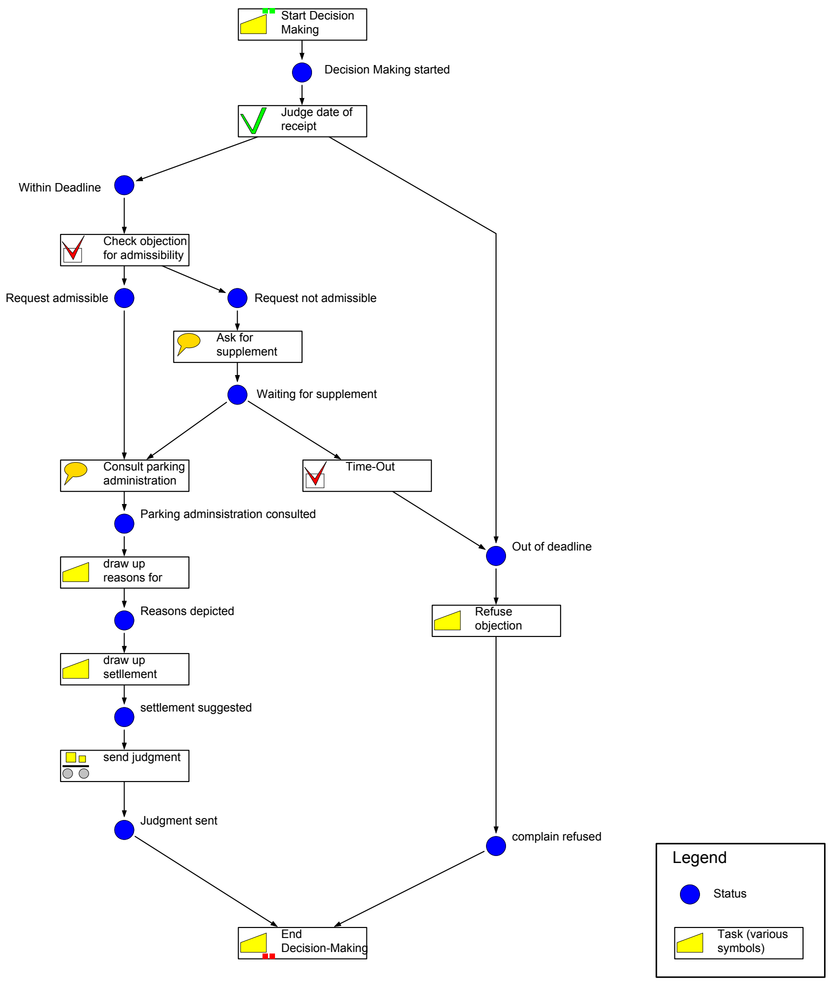

Fig. 1. An example decision-making process about objections against parking fines, modelled in Protos

During this research Pallas Athena made a set of reference process models and a set of adaptations of these models available to us. Combined, the sets contained more than 500 process models. Although guidelines how to model sound business processes in Protos and tools for verification exist [23,24,25], we discovered that most of these models do not conform to the guidelines. In addition, the models lack of data required for process simulations which also means that simulation [26] was hardly used, if at all. Thus, we can assume that the process designers were either unaware or not convinced of the value of sound models and simulation. Looking into the current simulation of Protos also we had to realize that it is unclear which of the parameters that can be specified in Protos are actually used for the simulation. For example, we discovered that in Protos a field for the number of resources required for the execution of a task exists, but the simulation always uses just a single resource and neglects this parameter.

Within this paper we will present two new tools enabling Protos users to test and validate their process models with the help of colored Petri nets (CPNs) [14]. Both tools are available for download from `http://www.florian-gottschalk.de/protos2cpn`.

First, we will depict a new way to simulate business processes modelled in Protos using CPN Tools [16, 27]. Nowadays CPN Tools is probably the most popular modelling and analysis environment for CPNs, used in more than 120 countries with more than 4500 licensees. It provides not only a nice way to visualize running processes but also extensive measurement and verification opportunities for concurrent systems. Within this research we developed a transformation from Protos models to CPNs, using the same data as the current Protos simulation. Using the simulation of CPN Tools we enable the unexperienced user to see directly in which order the process tasks are executed and what might go wrong in incorrect workflow models. In addition some basic statistics are provided to her. The advanced user will be able to add additional measurements to process models as well as she can see which of the Protos parameters are actually used during the simulation. The current Protos simulation is using a tool called ExSpect [4,26] which is based on another type of colored Petri nets. However, ExSpect does not allow for the easy creation of additional measurements. Its standard layout scheme, which is applied when loading a model, causes unacceptable delays when trying to inspect or interact with running processes. In addition, the development of ExSpect has stopped for some time already [12].

Second, we change the transformation in such a way that it creates a configurable process model from the Protos model. We developed configurable process models in our previous research as a general mechanism for process model adaptation [5,13]. When configuring a process, its unnecessary parts are eliminated, i.e. the possible process behavior is restricted [9,10,20,21]. Incorporating configuration options into the process model during the transformation creates for the first time a tool that enables users (1) to apply process configuration decisions on a process model without changing the model's structure, and (2) to test these decisions by direct interactions with a simulation model.

The tools are the result of a 6 months project in which the authors were involved at different stages and levels. Initial ideas for transforming Protos models into CPN models combined with the fundament for the tools were developed by one of the authors already before the project started. For the project these ideas were then combined with the idea to use the tool for configuration which required the evolvement of the initial ideas to a usable product.

The remainder of the paper is structured as follows: Section 2 presents the transformation from Protos models to CPNs. Section 3 contains a short introduction into configurable process models, a description how these ideas have been implemented in the CPNs derived in Section 2, and, based on four exemplary configuration patterns, an outlook on possible soundness issues caused by the process configuration. The paper concludes with a summary and an outlook on open issues.

2 Protos2CPN: From Protos Models to Colored Petri Nets

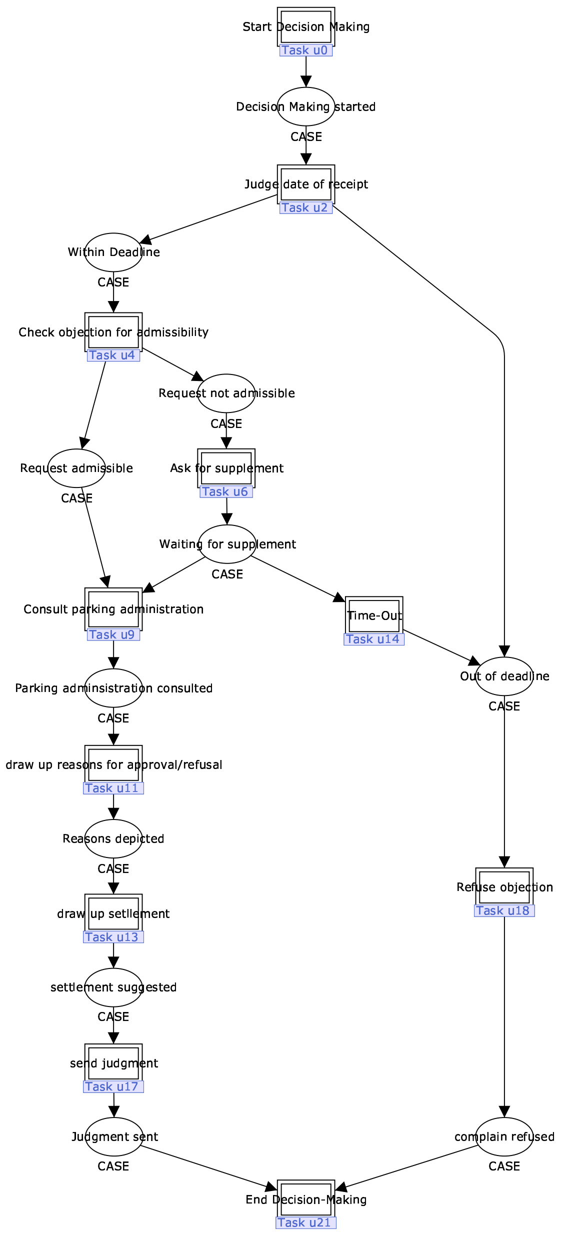

Fig. 2. The overall process model of the CPN (generated from the Protos process model depicted in Figure 1)

Basically Protos2CPN converts the data provided by Protos for the current simulation into a CPN, executable in CPN Tools. So far, CPN Tools provides no opportunities for importing other file formats than the CPN Tools XML-file format. It is therefore reasonable to use an XML export of Protos and transform this into the CPN Tools format by an XSL transformation. The current Protos simulation [26] is already using temporary stored XML files[2] for the communication between Protos and ExSpect. So we decided to "plug-in" in between using the same files for the generation of the CPN Tools files, especially as our main goal is the process simulation as well. The exported XML file includes a flattened process structure (i.e, the hierarchical structure of Protos models is reduced to a single level process model), the resource utilization of each task, and further statistical simulation parameters. It lacks of information about the layout of the process, but we aim at providing an automatic layout functionality. Currently, the models are created with a basic layout scheme that allows for an easy re-arranging of process elements, keeping the manual effort reasonable.

To enable the user to look at the CPN in the same way as to the Protos model, without the need for learning a new "complicated" modelling language [19], it is

[2] These XML files are generated when starting a simulation (see [26] for details), and should not be confused with the regular XML export of Protos which is accessible through the File/Report menu.

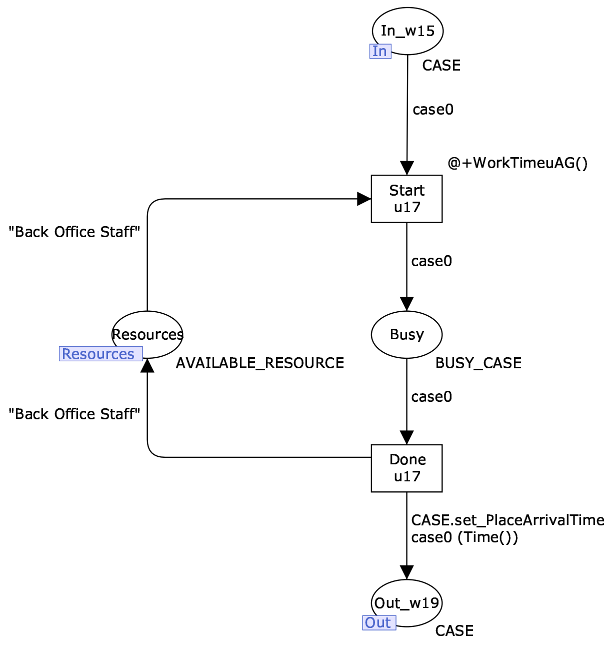

Fig. 3. Dedicated task page with information for simulation

our goal to transform the Protos models into CPN models that match the Protos model in look and structure as closely as possible. For that reason we decided that any information not depicted in the process view of Protos (cf. Figure 1) but maintained in property dialogues and needed for simulation must be depicted on separate sub-pages. The derived CPN model provides therefore two levels: First the overall process model, similar to the (flattened) process view of Protos, where every Protos status[3] is transformed into a CPN place and each Protos task is transformed to a CPN substitution transition[4] (cf. figures 1 and 2); and second the sub-pages of these substitution transitions representing an "execution layer" incorporating all data relevant for the simulation (cf. Figure 3).

The places in the overall process model are allowed to have tokens of the type `CASE`[5] depicting the details about the cases running through the process such as a case id, the start time of the process, and the arrival time of the process in the particular place:

```
colset CASE = record CaseID:INT *
                     ProcessArrivalTime:INT *
                     PlaceArrivalTime:INT;
```

[3] The Protos name for a place (e.g., a model object representing a channel or state) is status.

[4] Note, that in Protos it is possible to connect transitions directly to transitions or statuses directly to statuses whereas in (colored) Petri nets this is not. In this case additional auxiliary statuses or transitions are introduced into the CPN overall process model. This is already done by Protos when creating the XML export.

[5] In order to distinguish between color sets, labels and variables in the CPNs, all color sets are written in capital letters, all variables in lowercase letters, and all labels start with a capital letter.

Whenever a token is residing in one of the places of the overall process model it is waiting for execution by one of the subsequent transitions. All specifications of how and when a task can be performed by one of these transitions are "hidden" on the corresponding sub-page for the task, i.e. the transition depicted in the overall process model is just a substitution transition for the underlying sub-page which is named according to the task's unique ID.

Instead of the full task and status names, the sub-pages of the overall model use distinct IDs to refer to statuses or tasks. These IDs are provided in the Protos export, task IDs start with a "u", status IDs with a "w".

Each sub-page consists basically of two transitions: A `Start` and a `Done` transition (cf. Figure 3). The `Start` transition symbolizes the start of the task and is enabled by token(s) on its input place(s), i.e. by tokens arriving on the preceding places on the overall process model page. When the `Start` transition fires, it puts a token into a place `Busy`, symbolizing the lasting of the task. The duration of the task, i.e. the delay, is determined by the function `WorkTimeuXX()` which calculates the duration of the task according to the specification at the corresponding task in the Protos model (in the model `XX` is replaced with an automatically calculated code serving as a shortcut for the name of the particular task). By adding the delay to the token it is ensured that the token cannot be removed from the `Busy` place while the task lasts. Firing the `Done` transition depicts the completion of the task. It removes the token from the `Busy` place, and puts token(s) into the output place(s), i.e. in the succeeding place(s) within the overall process model. As such a token arrives in a new place of the overall process model, the place arrival time of the particular case is updated with the new actual point in time while the token is put into the succeeding place (function `CASE.set_PlaceArrivalTime`).

The number of input and output places depends on the number of incoming and outgoing arcs of the particular task within the Protos model. In the attributes of a task in Protos it can be specified if several incoming arcs depict either an AND-Join or an XOR-Join. Several outgoing arcs of a task can depict either an AND-split or an XOR-split. This might be confusing for Petri net users as the substitution transitions within the overall process model are looking like a standard Petri net AND-join/-split, but can represent both XOR and AND-joins/-splits. The exact behavior is only modelled within the corresponding CPN sub-page:

– In case of an AND-join each input place is connected to the `Start` transition. The case id's of incoming tokens from different arcs are synchronized by a guard attached to the `Start` transition. (see Figure 4).
– In case of an XOR-join, a `Start` transition is introduced for each input-place/arc. So a `Start` transition can fire and result in the busy state of the task when-

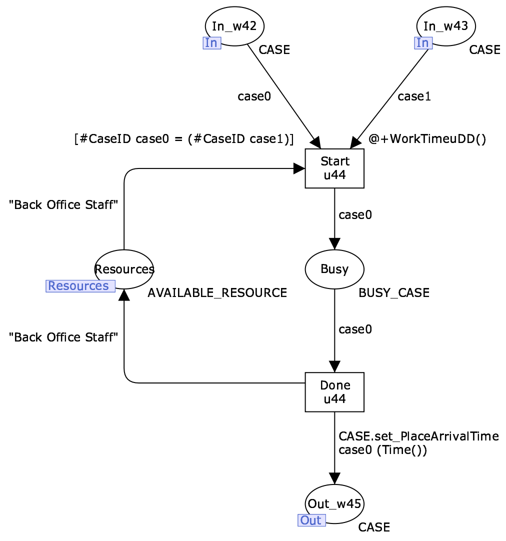

Fig. 4. The AND-Join modelled using a CPN sub-page: The transition `Start u21` needs a token with the same case id in both input places (`In_w19` and `In_w20`)

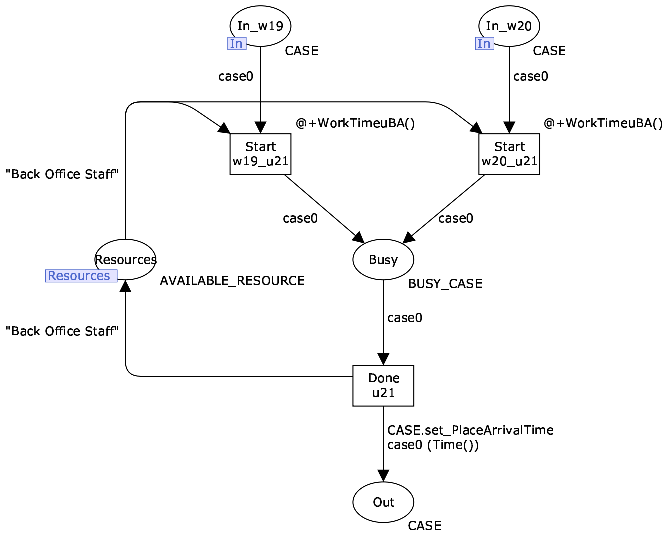

Fig. 5. XOR-Join: Each `Start` transition is enabled as soon as a token arrives in the corresponding `In` place and the resource "Back Office Staff" is available.

ever a token is placed into one of the input places (see Figure 5).

– In case of an AND-split the `Done` transition puts tokens into all output places (see Figure 6).
– In case of an XOR-split, a `Done` transition is introduced for each output place/arc (Figure 7). So only one `Done` transition can take the token out of the `Busy` place and fire by removing the token from the place `Busy`, releasing the resource and putting the case token into the corresponding `Out` place. The other `Done` transitions become disabled as soon as the first one fires as no token remains in the `Busy` place.

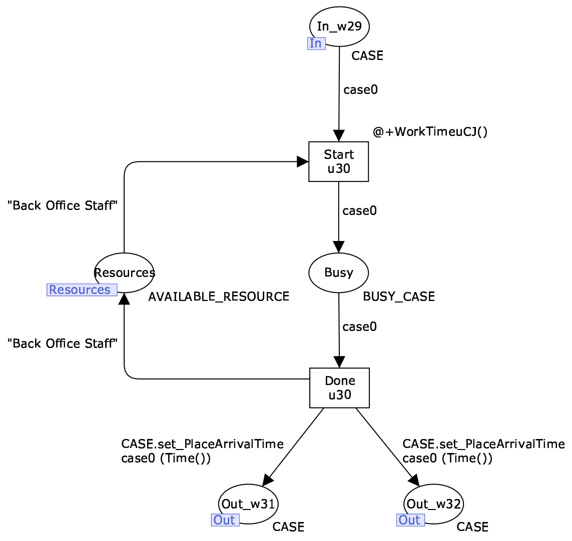

Fig. 6. AND-Split: The transition `Done u2` releases the resource "Back Office Staff" and forwards the case to both outgoing arcs by putting a case token into each of the two `Out` places.

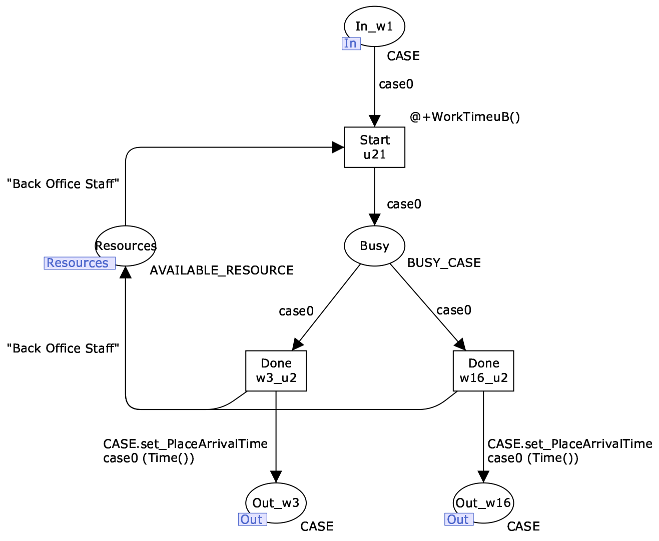

Fig. 7. XOR-Split: As soon as the case is not "busy" anymore, both `Done` transitions become enabled, but only one can fire.

In addition to the control flow behavior also resource utilization is depicted in the sub-pages. For that reason every sub-page contains a place `Resources`. All these resource places are members of a fusion set containing the available resources:

```
colset AVAILABLE_RESOURCE = STRING;
```

Transition `Start` can only fire if the required resources are available. The resource requirements for a task are specified on the arc from the place `Resources` to the transition `Start`. When the transition `Done` is fired, depicting the completion of the task, the previously occupied resources are released, i.e. the resources removed by the `Start` transition are put back into the `Resources` place. They can afterwards be used for other tasks.

Besides the overall process model and the sub-pages for the tasks, Protos2CPN creates a page `Arrival System`. It consists of a single transition spawning cases into the input place of the first Protos task based on some predefined arrival process, e.g. a Poisson arrival process using negative exponential inter arrival times. In this way it is ensured that the simulation is continuously fed with new cases.

Altogether the CPN model enables a step-by-step simulation of the Protos model, allowing a detailed analysis of diverse process behavior. In addition the monitoring-features of CPN Tools enable complex data collection while simulating processes. When transforming Protos models to CPN, Protos2CPN generates automatically three types of data collector monitors:

– A data collector monitor measuring the total flow time through the process per case,
– A data collector monitor measuring the resource availability/resource utilization over time per resource type, and
– A data collector monitor measuring the waiting time of cases per task (i.e. the time cases are waiting for execution by the particular task solely due to the lack of resources).

Normally the simulation would run forever as cases are continuously spawned into the system. To stop the simulation a breakpoint monitor is defined on the arrival system. By default it stops the simulation after 500 cases have been spawned into the system. This value can of course be adapted to individual requirements. Using the function `CPN'Replications.nreplications n` the performance of n replications of the simulation can be started. Thus, Protos2CPN provides this function for the execution of by default four replications on a dedicated `SimulationStart` page.

The automatically generated statistics can be used to find bottlenecks or other performance related problems of the model. When analyzing the data, keep however in mind that it might include a warm up period before reaching a steady state. Additional, more complex monitors can be added by the advanced user in CPN Tools. Users interested in details are referred to the monitoring help pages of CPN Tools [28].

3 Protos2C-CPN: Using CPN for Building Configurable Process Models

The second tool we developed is an extended variant of Protos2CPN allowing process model configuration within the generated model. For that reason we called it Protos2C-CPN where C-CPN stands for "configurable colored Petri net". When configuring a process model, e.g. the process model depicted in Figure 1, some of the tasks of the model are eliminated in such a way that they cannot be performed when the process is enacted.

CPN models created by Protos2C-CPN provide dedicated features for process model configuration. These features enable the user to adapt the model to individual requirements without changing the structure of the original reference model. Afterwards the configured model can either be tested on its feasibility, i.e. its soundness, or it can be simulated in the same way as the models generated by the Protos2CPN tool depicted in the previous section. By applying and simulating different configurations on the same process model their particular efficiency could even be compared.

To depict how Protos2C-CPN can be used in this context and also to show the limitations of Protos2C-CPN, this section is split into three parts. First we will give some background information on the ideas behind configuration of process models. Second we will explain how these ideas are incorporated into the CPN models. And third we will conclude this section with an outlook on possible limitations of the configuration approach introduced in this paper which is based on the analysis of four exemplary configuration decisions for the process model from Figure 1.

3.1 Configuring Process Models

Configuration is a mechanism for adapting process models that restricts the possible run-time behavior of a process [8,9]. As an example, removing an arbitrary task from the process model in figures 1 and 2 would be configuring of the process model. However, according to our definition of configuration, operations such as the adding of additional tasks or the renaming of model elements are not possible by means of configuration. Also note that not all configurations are sound/valid.

Based on concepts adopted from the inheritance of dynamic behavior [3,7], we identified two mechanisms for configuration of process models in previous research, called *blocking* and *hiding* [5,13]. Blocking refers to encapsulation. If a task in a process is blocked, it cannot be executed. The process will never reach a subsequent state and therefore all (potential) subsequent tasks will not be executed as well. If, e.g., the task `Time-Out` in Figure 1 (or Figure 2) is blocked the process will never reach the status `Out of deadline` from the place `Waiting for supplement` and thus also never execute the task `Refuse objection` after it has reached the place `Waiting for supplement`. Hiding corresponds to abstraction. If a task is hidden, its performance is not observable, i.e. it consumes neither time nor resources when it is executed. But the process flow continues afterwards and subsequent tasks will be performed. For that reason we also talk about a *silent task* or simply about *skipping the task*. If, e.g., the task `Draw up reasons for approval/refusal` in figures 1 and 2 is

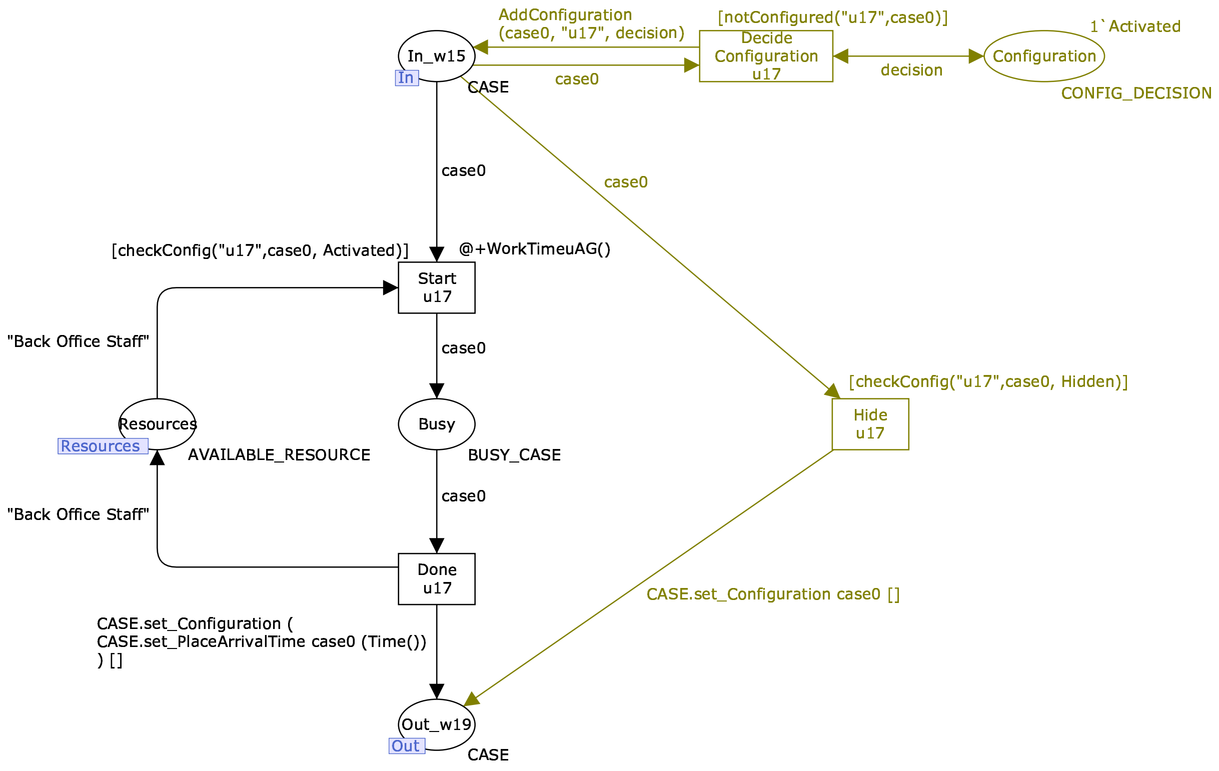

Fig. 8. A sub page of a configurable task

hidden, the task `draw up settlement` will be the next task after the parking administration was consulted. So, whenever a certain task should not be executed, but the process flow should be able to continue in this direction, the task must be hidden. If the process flow should not continue in this direction, the task must be blocked.

Configuration decisions about blocking or hiding of tasks are typically made before the execution of the process. However, sometimes not all information required to decide between activating, blocking, or hiding might or needs to be available in advance. For example, in certain cities an accelerated procedure which allows skipping the task `Draw up reasons for approval/refusal` might be applied whenever the fine has been imposed by police officers instead of traffic wardens. However, the information of who has imposed the fine is only found out when consulting the parking administration. Thus, the configuration decision if the task `Draw up reasons for approval/refusal` has to be executed must be transformed from a configuration decision into a run-time decision, and has to be made on a case-by-case basis during the process execution.

3.2 Configurable CPN

To cope with the two mechanisms for configuration, blocking and hiding, the models derived in Section 2 have to be extended with additional behavior. As process configuration is defined on a task level this can be done on the sub-page of each task. A task is activated if it is neither blocked nor hidden. This corresponds to the situation in ordinary, i.e. non-configurable, models. Within the CPNs generated by Protos2C-CPN these three configuration opportunities are distinguished by using the color set `CONFIG_DECISION`:

```
colset CONFIG_DECISION = with Activated |
                              Hidden |
                              Blocked;
```

The decision between activating, hiding, and blocking has to be made for each task individually. For that reason a place `Configuration` is added to each sub-page as depicted in the top-right of Figure 8. When configuring the task the default decision, i.e. the initial marking of the place `Configuration`, can be changed from `Activated` to `Hidden` or `Blocked`. By implementing the configuration decision as a marking of a place (instead of defining it, e.g., as a constant arc inscription) the configuration can be changed without changing the net structure.

Whenever a token arrives in an input place of a task, the transition `Decide Configuration` is enabled. When firing, this transition takes a configuration from the `Configuration` place (variable `decision`), combines it with the number of the particular task ("u17" in the example in Figure 8), and adds it to the list of configurations made for the corresponding case (function

```
fun AddConfiguration (case0:CASE,task:STRING,decision:CONFIG_DECISION) =
    {CaseID=(#CaseID case0),
     ProcessArrivalTime=(#ProcessArrivalTime case0),
     PlaceArrivalTime=(#PlaceArrivalTime case0),
     Configuration=({Task=task,Configuration=decision} :: (#Configuration case0))
    };

fun checkConfig (task:STRING, case0:CASE, config_decision:CONFIG_DECISION) =
    List.exists
        (fn a => (#Task a) = task andalso (#Configuration a) = config_decision) (#Configuration case0);

fun notConfigured (task:STRING, case0:CASE) =
    not(List.exists (fn a => (#Task a) = task) (#Configuration case0));
```

Fig. 9. The functions for adding and checking configuration decisions

`AddConfiguration`, see Figure 9). This list is an additional attribute to the color set `CASE`:

```
colset TASK_CONFIGURATION =
    record Task:STRING *
           Configuration:CONFIG_DECISION;
colset TASK_CONFIGURATIONS =
    list TASK_CONFIGURATION;
colset CASE =
    record CaseID:INT *
           ProcessArrivalTime:INT *
           PlaceArrivalTime:INT *
           Configuration:TASK_CONFIGURATIONS;
```

If the configuration decision is not clear during the phase of process configuration, tokens for all possible configuration decisions can be put into the `Configuration` place at the same time. The decision will then be made at runtime on a case-by-case basis. Then the transition `Decide Configuration` "selects" one of the configuration tokens before the task can be started. The function `notConfigured` (see Figure 9) in the guard of the transition ensures that for each arriving case this decision can only be made once per task by checking the list of already added configuration decisions.

The guard of the `Start` transition ensures that the task can only be started in case it is activated, i.e. if an element of the list of configurations combines the task with the decision `Activated`. For this, the function `checkConfig` (see Figure 9) uses the Standard ML function `List.exists` which returns true if its first parameter returns true for any of the elements in `case0`'s list of configurations.

In case the task is hidden, it is required to bypass the `Busy` place. This is done by an additional `Hide` transition, connecting the input place directly with the output place, without using any resources. The `Hide` transition is only enabled if the case token contains a `Hidden` decision for the particular task which is again enforced by the guard of the transition.

If the case is blocked for the particular task, no further behavior is allowed within this task. For that reason no transition on the sub-page is able to remove a token from the input place which is blocked for the actual task. This needs to be done by another task.

If the corresponding Protos task is an XOR-join, multiple `Hide` transitions and multiple `Decide Configuration` transitions will be introduced, similar to the multiple `Start` transitions introduced in Section 2. As the `Hide` transitions combine the `Start`- and the `Done` transitions, additional `Hide` transitions must also be introduced in case of an XOR-split. Then each `Hide` transition combines one of the alternative input places with one of the alternative output places. So the maximum amount of `Hide` transitions (in case of an XOR-join and an XOR-split) is "incoming arcs of the Protos task" times "outgoing arcs of the Protos task". As the embedding is analogous to the description and the figures 5 and 7 of the non-configurable model in Section 2, we omit further figures at this point.

As the state of the token has changed after the execution of the task, all configuration decisions which have been made already, must be re-evaluated afterwards. For that reason the list of configuration decisions is set back to the empty list by the function `CASE.set_Configuration` before the case token leaves a sub-page via the `Out`-place. If the task was activated and therefore executed, this is done in addition to the update of the place arrival time. Note, that the task requires no time if it was hidden, and thus it is then not required to update the token's place arrival time.

3.3 Soundness Analysis and Limitations of Configured Process Models

After implementing configurable process models and configuring them, we are now able to test the configured process models on their feasibility, i.e. their soundness, and on their performance. The notion of sound workflow nets [1] expresses the minimal requirements any workflow (and therefore also any executable process model) should satisfy. Simulation allows for the evaluation of various configurations in the same manner as described in the end of Section 2. By simulating different configurations the results can also be compared. In this paper we will, however, only focus on the testing of soundness as the basic prerequisite for every process model.

A workflow net is sound if it satisfies the following conditions:

1. *Option to complete*: There should always be an option to complete a case that is handled according to the process definition.
2. *Proper completion*: A workflow should never signal completion of a case while there is work in progress on this case.
3. *No dead tasks*: For every task there should at least be a chance that it is executed, i.e. for each task there should be at least one execution sequence that includes the task.

Although, theoretically only the configured models need to be sound, we require within this approach that the reference model itself is sound as well. In this way we ensure that every model element can be part of a sound process model. Otherwise these elements would have to be blocked in all configurations and would therefore be superfluous. The soundness of the reference model can be tested in Protos using Woflan [25]. After configuring the reference model we can use CPN Tools' state space analysis tool to check how far the soundness conditions are satisfied for a configured model. Details on how to perform a state space analysis with CPN Tools can be found in the CPN Tools State Space Manual [15]. In the following we only explain possible applications of its results.

As the size of the state space may grow exponentially (or worse) with the size of the process model, the model's complexity and its initial marking are reduced for soundness testing as follows:

- The process model is reduced to a single case as cases are handled independently and hence interactions among cases cannot invalidate soundness.
- All timing aspects are neglected as the untimed net includes every order of task execution.
- All resource requirements are neglected as soundness is here purely defined on the control-flow perspective (resource requirements do not influence soundness as long as resources are not assigned incrementally and as long as no task requires more resources of a certain type than resources of this type are in total defined).

To test the first condition "Option to complete" our approach requires to check the list of dead markings, i.e. the possible markings of the net in which no further behavior is possible: If in such a marking a token remains in a place which is not the "final place" of the process, the condition is violated. In a dead marking, tokens cannot remain in the busy places of sub-pages as the `Done` transitions will always be enabled after a `Start` transition has fired. It is therefore possible to test if a dead marking exists that marks a place of the overall process model (Figure 2). Such a case will never be completed as the Protos model (and therefore also the overall process model of the CPN) completes with the execution of the last task[6]. Then the condition is violated and the configured net is not sound. However, this approach does not cover livelocks: If the process models contains loops which can be entered by tokens, but all tasks allowing to exit this loop are blocked this check will fail because then the tokens will "circle" without reaching a dead state or completing. Such a situation can only be detected by analyzing strongly connected components. Any strongly connected component that has no outgoing edge to any other strongly connected component should not contain a state with a place on the overall process model page marked. Otherwise the option to complete is violated and the configured net is not sound. Note that there can not be a cyclic path between connected components, as this would result in one big connected component. As a result, the system will always be able to reach a strongly connected component that has no outgoing edge to any other strongly connected component.

It is not required to explicitly test the second condition to verify that the the reference model is sound. Configuration only restricts the possible process behavior. For that reason configured process variants cannot produce tokens which are not produced by the complete reference model; i.e., it is impossible that new tokens which indicate the completion of the workflow are generated. Within the reference model the proper completion might, e.g., be ensured by AND-joins in the termination task. Such task behavior is kept in the configured process model even if the task itself is hidden. So the completion of a case in the configured process can only be observed if the same conditions for completion are satisfied as required by the reference model. If this is impossible, tokens will remain in the process model, which is detected by the test for the first condition.

A task of a process is dead if the `Start` transition on the task's sub-page is a dead transition. Dead tasks are indeed not desirable in a sound workflow net. However when configuring a process, i.e. restricting its possible behavior, dead tasks may be desirable. The dead tasks are the unnecessary tasks that can be removed from the configured net. When analyzing the configured net it is therefore required to check if the dead tasks are those tasks which were intended to be removed.

In the following we will use the decision-making process from Figure 2 to discuss four example configurations of this process. Making use of the results provided by the

[6] On the process level the last (termination) task is not connected to any subsequent status/place which could be marked by it, i.e. all tokens are "consumed" by this last task. For that reason a properly completed case leaves no tokens behind on the overall process model page.

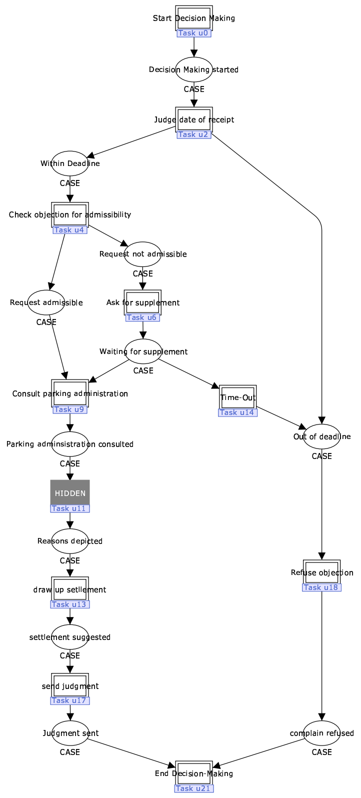

Fig. 10. Task `draw up reasons for approval/refusal` hidden (grey)

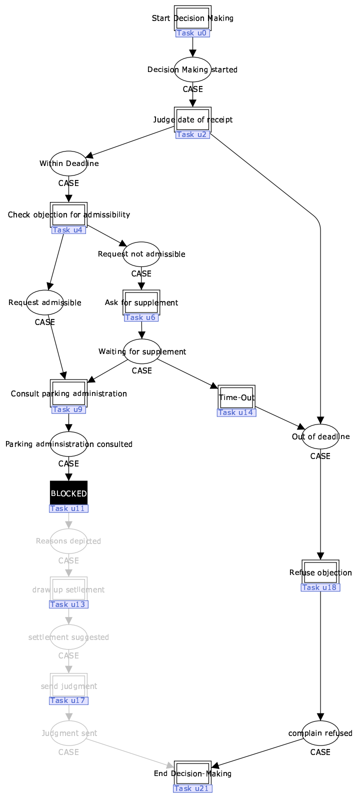

Fig. 11. Task `draw up reasons for approval/refusal` blocked (black)

state space analysis, we will depict and analyze the purpose and sense of the configuration decisions of blocking and hiding for the selected tasks in the particular context. This analysis is far from complete as it is based on examples in which we address only selected workflow patterns. For example, it does not contain any loops, and thus does not contain the risk for livelocks. It is included to demonstrate the existence of certain process configuration patterns.

Configuring tasks in sequence patterns

The task `draw up reasons for approval/refusal` is located in a sequence of tasks between the task `Consult parking administration` and the task `draw up settlement`. In some municipalities it might be sufficient to draw up the settlement without explicitly drawing up reasons. In this case a single token "Hidden" is placed in the configuration place of task `draw up reasons for approval/refusal` (task ID: u11, see Figure 10). If we run a state space analysis the corresponding report contains only `Task_u11'Start_u11` and `Task_u11'Done_u11` as dead transitions (besides all the other `Hide` transitions) which is exactly what we wanted to achieve: the task is never executed, but the subsequent tasks are executed.

If the task is configured as blocked (see Figure 11) the state space analysis lists further dead transitions:

```
Task_u11'Done_u11
Task_u17'Done_u17
Task_u11'Hide_u11
Task_u17'Hide_u17
Task_u11'Start_u11
Task_u17'Start_u17
Task_u13'Decide_Configuration_u13
Task_u21'Decide_Configuration_w19_u21
Task_u13'Done_u13
Task_u21'Hide_w19_u21
Task_u13'Hide_u13
Task_u21'Start_w19_u21
Task_u13'Start_u13
Task_u17'Decide_Configuration_u17
[...]
```

Neither the Task `draw up settlement` (task ID: u13) nor the task `send judgement` (task ID: u17) will ever be started. It is even never needed to decide its configuration. This means tokens will never be in the place `Reasons depicted` or `settlement suggested` which is also indicated by the upper and lower bounds of these places in the state space report which are 0. Also the task `End Decision-Making` (task ID: u21) will never be exe-

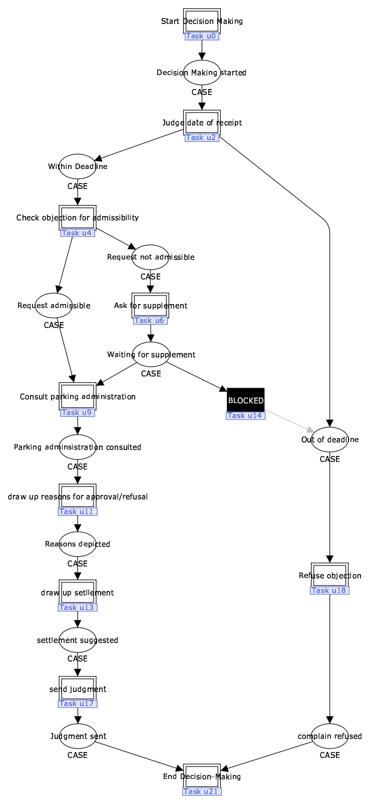

Fig. 12. Task `Time-Out` blocked (black)

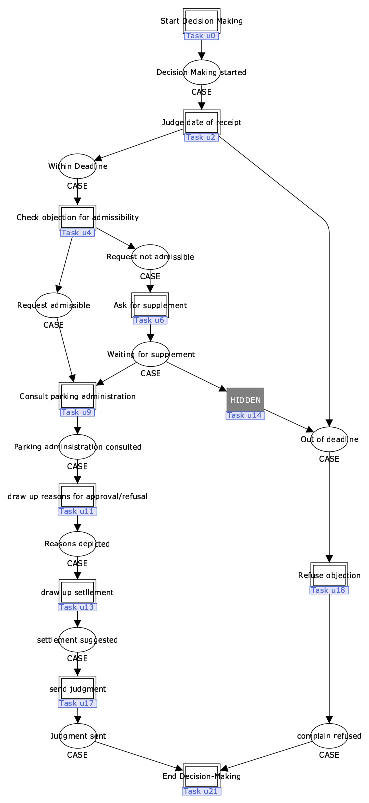

Fig. 13. Task `Time-Out` hidden (grey)

cuted from place `Judgment send` (place ID: w19) which is indicated by the last three dead transitions.

These results of the state space analysis are not surprising as it is exactly what was intended when blocking the task. However, the configured net is not sound: In some of the dead markings a token, i.e. the case, remains in the place `Parking administration consulted` which is not the final place of the process. As depicted this is not allowed in a sound workflow net, which means that the net would remain incorrect even after removing all dead model parts. We can conclude that the blocking of a task in a sequence causes problems.

Configuring dummy tasks in deferred choice patterns

The task `Time-Out` is executed when the supplement was not received within a certain period of time. This means there is a race condition between the timer triggering the `time-out` and the receival of the supplement triggering the task `Consult parking administration`. The decision which of the two tasks is executed is postponed until the execution of one of the tasks starts. Therefore this situation is also called a deferred choice. If the municipality decides that cases cannot time-out, the task `Time-Out` has to be blocked (see Figure 12). Then its `Start` and `Done` transitions are listed as dead in the corresponding state-space report. However, the state space analysis reports no dead states with tokens remaining in places other than the final place. That means in case of the construct of a deferred choice, it might be possible to block a task without creating a deadlock.

If the task `Time-Out` is hidden (see Figure 13), it will never start nor finish but its `Hide` transition will fire, and the case reaches the `Out of deadline` place. This seems to be fine from a syntactical perspective. However when executing the process, it becomes obvious that the behavior of the process practically conforms to the behavior of the activated `Time-Out` task. This phenomenon occurs due to the fact that the `Time-Out` task can be seen as a dummy task which is a task not corresponding to the execution of any work but introduced for changing the state of a process model, e.g. triggered by an external event. As there is no output produced, such dummy tasks are also called silent tasks or silent transitions. The hiding of such a task is questionable because in this case the effect of hiding and activating is quasi-identical.

Configuring tasks with interdependencies with other tasks

If a municipality does not want to ask for supplements in case a request is not admissible, one could think of blocking the task `Ask for supplement` (see Figure 14). But then the municipality would end-up with cases lost in the place `Request not admissible`, never reaching the

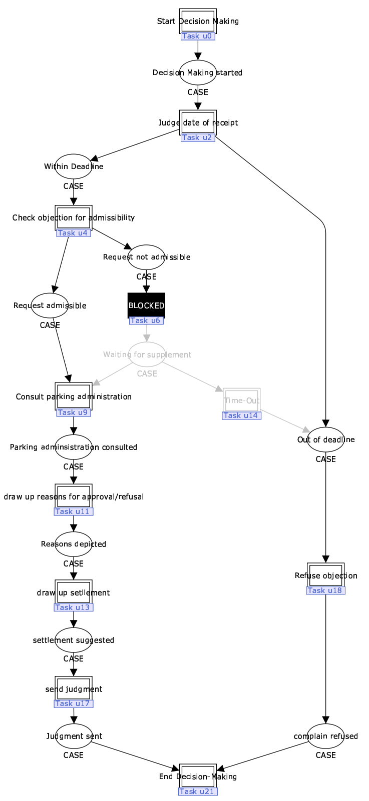

Fig. 14. Task `Ask for supplement` blocked (black): Lost tokens may remain in the place `Request not admissible`

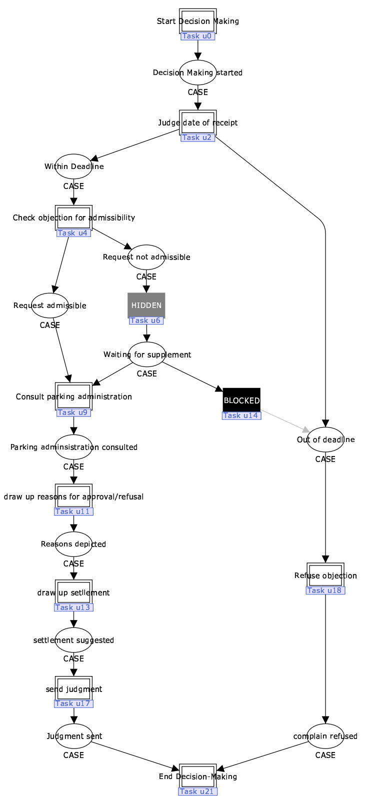

Fig. 15. To avoid asking for supplements, not only the Task `Ask for supplement` is hidden (grey), but also task `Time-Out` must be blocked (black)

final task. So, the other option is to hide the task `Ask for supplement` which results in another issue: Non-admissible requests might time-out while waiting for an action by the municipality. Formally this is not a problem, but content-wise it could be unintended behavior. To resolve this issue and create a valid configuration, the dummy task `Time-Out` must be blocked additionally whenever the task `Ask for supplement` is hidden (see Figure 15).

This means that configuration decisions are not always independent of each other. Whenever a configuration decision eliminates a task's execution, it must thus be checked, if the performance of other tasks depends on the not-performed task. If this is the case, also the execution of these dependent tasks must be eliminated by further configuration decisions.

Configuring tasks in explicit choice patterns

When executing the task `Judge date of receipt` an explicit choice how the process will continue is made: If the complaint was received within the deadline, it will be checked for its admissibility, whereas in case it arrives too late, it will be refused. But maybe some municipalities want to be less restrictive with the initial deadlines and consider all objections as being received within the deadline. So all objections must be checked for their admissibility. However, neither hiding nor blocking of the task `Judge date of receipt` helps here. If it is blocked the process will never go beyond this task. If it is hidden, tokens can still be placed into the `Out of deadline` place.

In the previous scenario we could achieve the desired behavior by hiding one (`Ask for supplement`) and blocking another task (`Time-Out`). But also this approach is impossible to apply as neither hiding nor blocking of task `Refuse objection` can prevent that tokens are put into the place `Out of deadline` and as soon as a token is in this place it cannot be checked for its admissibility anymore.

The only chance of enforcing the desired behavior is, to go to a lower level, i.e., to have a look at the implementation of the choice on the task page. In a standard Petri net an explicit choice can only be modelled by a deferred choice with subsequent silent transitions. In our implementation of the XOR-split, these silent transitions are the multiple `Done` transitions. Only when explicitly blocking the particular `Done` transition on this task-level (see Figure 16), it is possible to restrict the process model to the desired behavior[7]. So within a sound pro-

[7] The configuration of `Done` transitions is not yet implemented in the Protos2C-CPN transformation.

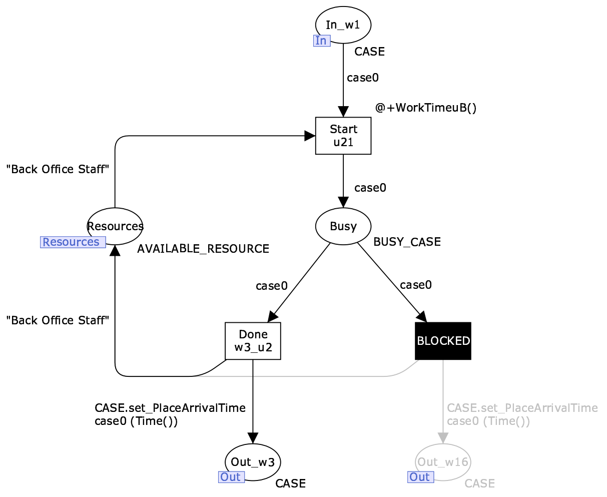

Fig. 16. `Done` transition blocked (black) in the sub-page of the **`Judge date of receipt`** task: The task can only exit via the left path.

cess model the outcome of an explicit choice cannot be restricted, i.e., configured, at the process level, but on lower implementation levels.

In general, the analysis of these four configuration scenarios with both the state space analysis and the simulation facilities of CPN Tools demonstrated the need for both of the two configuration mechanisms of blocking and hiding, as well as the need to implement them not only in a straightforward, but also in a creative way. Taking this into consideration, the examples also hinted on the power of these mechanisms.

4 Conclusions

By analyzing a set of reference models designed using the business process modelling language of Protos, we discovered that these models do not conform to well defined soundness criteria which also prevents the meaningful use of Protos' simulation features. The main reasons for this are that the developers of the models either see no value in the Protos simulation, or they are not aware of its value. We also realized that it is unclear which of the parameters that can be specified in Protos are actually used in the Protos simulation. To improve the value of simulation in this context, we developed the Protos2CPN transformation which allows the simulation of Protos models in CPN Tools. The simulation of Protos models in CPN Tools makes the running process visible by depicting the moving cases as tokens within the process model. It therefore allows for a detailed inspection of the running process. In addition, the monitoring features of CPN Tools enable the generation of comprehensive statistics which can serve as a basis for complex decisions. The models created by our Protos2CPN transformation already include some basic measurements which can be extended by experienced users.

In a second step we developed Protos2C-CPN. As far as we know, this was the first implemented tool offering explicit support for reference model adaptation by adding standard configuration features to the tasks in the reference models. These features permit the restriction of the possible behavior of the reference model directly in the model without changing its net structure. The simulation features of CPN Tools allow for performance testing and comparison of different process configurations. By making use of CPN Tools' state space analysis feature, we were able to test exemplary configurations on their feasibility in sound process models, but also realized that certain configurations are undesirable in specific contexts.

It might be possible to resolve such issues by looking at lower modelling levels. To explore this further, we plan to analyze configuration decision in the context of the workflow patterns [6]. We assume that by analyzing all workflow patterns on their configurability aspects, we can develop a set of configuration patterns, i.e. we will be able to generalize the discussion on the exemplary configuration scenarios. If such patterns are available, we could develop an improved version of Protos2C-CPN which might even be able to transform the configured model back into an ordinary process model which does not contain the configuration features anymore.

The idea of performing the transformation purely with XSL transformations proved feasible but far more complex than initially thought. To understand the source-code of the transformation deep knowledge of XSL transformations as well as of the XML definitions of both languages is needed. We are sure that using, e.g., XML facades in traditional programming languages as Java would help making the tool's code far better readable and manageable. Thus, before developing a new major version of the transformation, we will probably go through a deeper evaluation to see if it is worth to change the current approach.

Acknowledgements

The authors would like to thank Kurt Jensen and Lisa Wells for their continuous feedback and support related to the use of CPN Tools, as well as Pallas Athena for providing the Protos reference models.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig,

and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.

3. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, January 2002.
4. W.M.P. van der Aalst, P.J.N. de Crom, R.R.H.M.J. Goverde, K.M. van Hee, W.J. Hofman, H.A. Reijers, and R.A. van der Toorn. ExSpect 6.4 An Executable Specification Tool for Hierarchical Colored Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 455–464, Berlin, Germany, June 2000. Springer.
5. W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable Process Models as a Basis for Reference Modeling. In C. Bussler and A. Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 512–518. Springer Verlag, February 2006.
6. 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.
7. T. Basten and W.M.P. van der Aalst. Inheritance of behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
8. J. Becker, P. Delfmann, A. Dreiling, R. Knackstedt, and D. Kuropka. Configurative Process Modeling – Outlining an Approach to increased Business Process Model Usability. In *Proceedings of the 15th IRMA International Conference*, New Orleans, 2004. Gabler.
9. J. Becker, P. Delfmann, and R. Knackstedt. Konstruktion von Referenzmodellierungssprachen: Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle (in German). *Wirtschaftsinformatik*, 46(4):251–264, 2004.
10. J. vom Brocke and C. Buddendick. Konstruktionstechniken für die Referenzmodellierung (in German). In J. Becker and P. Delfmann, editors, *Referenzmodellierung. Grundlagen, Techniken und domänenbezogene Anwendung, also Proceedings of the 8th Fachtagung Referenzmodellierung*, pages 19–48, Heidelberg, 2004.
11. T. Curran, G. Keller, and A. Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Prentice Hall, Upper Saddle River, NJ, USA, 1998.
12. Deloitte & Touche Bakkenist. ExSpect Home Page. http://www.exspect.com.
13. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Configurable Process Models – A Foundational Approach. In J. Becker and P. Delfmann, editors, *Reference Modeling. Efficient Information Systems Design Through Reuse of Information Models*, pages 59–78. Springer, July 2007.
14. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
15. K. Jensen, S. Christensen, and L. M. Kristensen. *CPN Tools State Space Manual.* Aarhus, 2006.
16. Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3–4):213–254, June 2007.
17. R.J. Paul, G.M. Giaglis, and V. Hlupic. Simulation of business processes. *The American Behavioral Scientist*, 42(10):1551–1576, August 1999.
18. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, March 2007.
19. K. Sarshar and P. Loos. Comparing the Control-Flow of EPC and Petri Net from the End-User Perspective. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*, pages 434–439, Nancy, France, September 2005. Springer-Verlag.
20. R. Schütte. *Grundsätze ordnungsmäßiger Referenzmodellierung – Konstruktion konfigurations- und anpassungsorientierter Modelle (in German).* Gabler, Wiesbaden, 1998.
21. A. Schwegmann. *Objektorientierte Referenzmodellierung: theoretische Grundlagen und praktische Anwendung (in German).* Gabler, Wiesbaden, 1999.
22. A. Sharp and P. McDermott. *Workflow Modeling: Tools for Process Improvement and Application Development.* Artech House Publishers, Norwood, MA, 2001.
23. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan Home Page, Eindhoven University of Technology, Eindhoven, The Netherlands. `http://is.tm.tue.nl/research/woflan`.
24. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer-Verlag, Berlin, 2000.
25. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
26. H.M.W. Verbeek, M. van Hattem, H.A. Reijers, and W. de Munk. Protos 7.0: Simulation Made Accessible. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005: 26th International Conference (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 465–474, Miami, USA, June 2005. Springer-Verlag.
27. A. Vinter Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In W.M.P. van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer Verlag, June 2003.
28. L. Wells. Monitoring a CP-net. `http://wiki.daimi.au.dk/cpntools-help/monitoring_a_cp-net.wiki`.