Let's Go All the Way: From Requirements via Colored Workflow Nets to a BPEL Implementation of a New Bank System

W.M.P. van der Aalst[1,2], J.B. Jørgensen[2], and K.B. Lassen[2]

[1] Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. w.m.p.v.d.aalst@tm.tue.nl
[2] Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark. jbj@daimi.au.dk, krell@daimi.au.dk

Abstract. This paper describes use of the formal modeling language *Colored Petri Nets (CPNs)* in the development of a new bank system. As a basis for the paper, we present a *requirements model*, in the form of a CPN, which describes a new bank work process that must be supported by the new system. This model has been used to specify, validate, and elicit user requirements. The contribution of this paper is to describe two translation steps that go from the requirements CPN to an implementation of the new system. In the first translation step, a *workflow model* is derived from the requirements model. This model is represented in terms of a so-called *Colored Workflow Net (CWN)*, which is a generalization of the classical workflow nets to CPN. In the second translation step, the CWN is translated into implementation code. The target implementation language is BPEL4WS deployed in the context of IBM WebSphere. A semi-automatic translation of the workflow model to BPEL4WS is possible because of the structural requirements imposed on CWNs.

Key words: Business Process Management, Workflow Management, BPEL4WS, Colored Petri Nets.

1 Introduction

Bankdata is a Danish company that is currently developing a new system called the *Adviser Portal (AP)*. AP has been bought by 15 Danish banks and will be used by thousands of bank advisers in hundreds of bank branches. The scope of the system is to support advising private customers and small businesses. The total development effort is 15 developers over a period three years. The first version is planned to become operational in September 2005.

The main goal of AP is to increase the efficiency and quality of bank advisers' work. Currently, prior to the deployment of AP, the advisers in Bankdata's customer banks often need information, which is scattered over many places: in different IT systems, on paper sheets in binders or in piles on a desk, on post-it notes, or even only in the minds of advisers. This hampers both efficiency

and quality; it is time-consuming to search for information, and an adviser may, e.g., sometimes forget to call a customer when she has promised to do so. The scattering of information makes it difficult for an adviser to get an overview, both of her own current and future tasks, and of the information pertaining to a particular task. Moreover, it makes it difficult for the bank, as an organization, to coordinate, distribute, and plan work.

Problems like these are mainly caused by the nature of the bank advisers work processes. To overcome the problems, the AP system will represent a change in perspective for Bankdata's software development. Previously, Bankdata focused on the development of traditional data-centric systems. For the new AP system, Bankdata uses a process-centric approach: In the new system, there is more focus on the work processes that must be supported than there has been previously. Thus, a workflow management system is a central component of AP.

Our focus in this paper is on AP's support of one specific work process: The process describing how bank advisers give advice to customers enquiring about getting a so-called *blanc loan*. A blanc loan is a simple type of loan, which can be granted without requiring the customer to provide any security. This is in contrast to, e.g., mortgage credits and car loans. Blanc loans are typically used for consumption purposes like travels, weddings, and gifts. They constitute a relatively high risk for the banks and have a correspondingly high interest rate.

We will describe the use of the formal modeling language *Colored Petri Nets (CPNs)* [16, 20] in the development of AP. First, CPN has been used as a vehicle for requirements engineering for AP. This involved using a *requirements model* in the form of a CPN as the core ingredient of an *Executable Use Case (EUC)* [17] to describe new work processes and their proposed computer support. This has been a means to specify, validate, and elicit requirements in a number of workshops with future users and systems analysts from Bankdata. We will present the *Requirements CPN (RCPN)*, and we will briefly outline how it has been used. However, the main focus of this paper is on two translation steps taken to close the gap between the requirements model and the implementation of the new system. The first step translates the RCPN into a *workflow model* in the form of a *Colored Workflow Net (CWN)*, a new class of Petri nets that we will introduce. The second step translates the CWN into the chosen implementation language, which is *Business Process Execution Language for Web Services (BPEL4WS)* [5]. (In this paper we will simply use "BPEL" to refer to this de-facto standard.)

The CPNs we present in this paper are created using *CPN Tools*, a graphical environment to model, enact and analyze CPNs. (CPN Tools can be downloaded from `www.daimi.au.dk/CPNtools/`.)

This paper is structured as follows. Section 2 introduces the overall approach, including the two translation steps in focus. Section 3 presents the requirements model. Section 4 introduces the CWN modeling language and describes how the RCPN is translated into a workflow model in the form of a CWN. Section 5 discusses how the CWN is translated into BPEL. Section 6 discusses related work. Section 7 concludes the paper by summarizing the main results and discussing future work.

2 Overall Approach

Figure 1 summarizes the overall approach. Translation *T0* creates a model of the real world, in this case the processes and people within banks. The result of *T0* is the *requirements model*. In the technical report [18], we describe how *T0* is made in cooperation between users and Bankdata analysts, and how the requirements model is used in an iterative, prototyping fashion; it is constructed based on informal textual descriptions and diagrams and has served as an engine to drive a graphical animation.

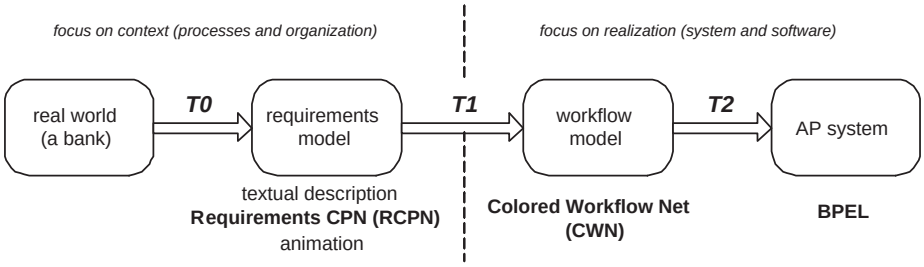

Fig. 1. Overview of the different models and translations between these models.

Translation *T1* derives the *workflow model* from the requirements model. The latter includes both: (1) actions that are to remain manual, when the new system is deployed; (2) actions that will be supported by the new system in interaction with human users; (3) actions that are to be fully automated by the new system. The workflow model includes only actions in categories (2) and (3). Moreover, the workflow model adds more details. The requirements model uses the CPN language in an unrestricted manner, e.g., tokens, places, and transitions may represent any entity deemed to be relevant. In the workflow model, we restrict ourselves to use only concepts and entities, which are common in workflow languages. More specifically, we propose *Colored Workflow Nets (CWNs)* as the language for making workflow models. A CWN is a CPN model restricted to the workflow domain and can be seen a high-level version of the traditional *Workflow Nets* (WF-nets) [1].

Translation *T2* goes from the CWN into skeleton code for the chosen implementation platform. AP is implemented using the IBM WebSphere platform; IBM WebSphere includes the workflow management tool IBM Process Choreographer, which will be used to orchestrate some of the work processes that are currently carried out manually in the banks. IBM Process Choreographer uses BPEL. Therefore, translation *T2* translates the CWN into BPEL.

As shown in Figure 1 by a dashed line, translation *T1* represents a shift in focus. Left of *T1*, the focus is on the *context* (processes and organization). Right of *T1*, the focus is on the *realization* (system and software). The use of CPN as a common language for both the requirements model and the workflow model

provides a natural link of these two views. This facilitates a smooth transition and is an attempt to avoid the classical "disconnect" between business processes and IT.

All translations *T0*, *T1* and *T2* in Figure 1 are done manually for the AP system in consideration. Both *T0* and *T1* are likely to remain manual given their characteristics; they inherently involve human analysis, decisions, and agreements between stakeholders. Translation *T2*, however, can be supported using a computer tool that generates template code for the chosen implementation platform. We have developed a systematic approach to transform a CWN into BPEL code. This approach is semi-automatic, i.e., the template code is generated on the basis of the structure of the underlying workflow net, as described in the technical report [4].

The translations we present should be seen as a proof-of-concept: They do not yield a full implementation of the AP system, but they merely demonstrate the viability of our approach.

3 Requirements Model

In this section, we first present the requirements model, i.e., the RCPN. The RCPN has been used as an ingredient of an Executable Use Case (EUC) [17], which support specification, validation, and elicitation of requirements. EUCs spur communication between stakeholders and can be used to narrow the gap between informal ideas about requirements and the formalization that eventually emerges when a system is implemented. An EUC may be seen as a *context-descriptive prototype* [6]. In this way, the RCPN has played a similar role as a high-fidelity prototype implemented in a programming language.

The use of EUCs in the development of AP is described in [18] (this is translation T0 of Figure 1). In the present paper, we merely give an impression of the EUC by showing the animation, which is part of it. Figure 2 mimics a situation in a bank in which there are two advisers, Ann and Bill, their manager Mr. Banks, and one customer, Mr. Smith. The circles represent blanc loan enquiries. The animation user can play with the animation and try out various scenarios and carry out experiments. Note that the animation is constructed using the animation facilities offered on top of CPN Tools and that the animated objects interact directly with the running CPN model.

We now present the RCPN; an extract is shown in Figure 3. At the same time, we also give an informal primer to CPN, which allows the reader to understand the CPNs in general terms. The primer does not account for all the technicalities of CPN; its purpose is only to provide an overall impression of the most basic concepts of the CPN language. For a more thorough introduction, the reader is referred to [16, 20].

A CPN describes a system's states and actions. The state of a system is modeled by *places*, drawn as ellipses. Places can hold *tokens*, which have different types or colors in CPN jargon. These types are a subset of the data types in Standard ML such as the primitive types integer and string and compositional

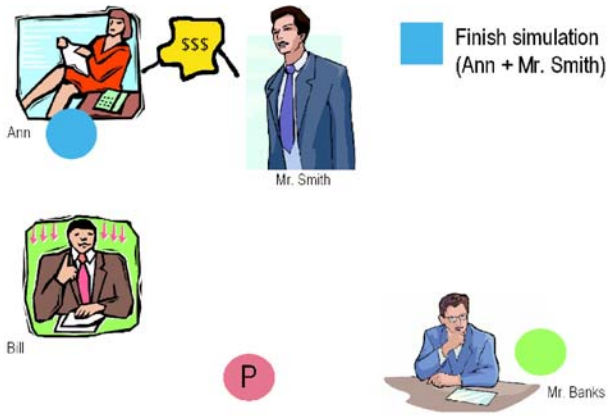

Fig. 2. Snapshot of graphical animation.

types such as tuple, list and record. Each place can hold tokens of a certain type. Usually the type of a place is written in capital letters close to it. All places in this CPN model have the type `LOAN`. Figure 4 shows the definition of `LOAN`.

`LOAN` is a record type; it consists of a *caseid* for separating different loan case instances, a *customer*, a *responsible* person (either an adviser or a manager), the *status* of the loan (ongoing, grant, etc.), the *amount* to loan, the *monthlyFee* to pay for the loan with a given loan setup, the *interestRate*, the *duration* for paying back the loan, the *purpose* for loaning the money and the *account* to put the money into.

A CPN's actions are represented by *transitions*, which are drawn as rectangles. *Arcs* connect transitions and places. An arc can only connect a transition with a place or vice versa; it is not possible to e.g. connect two places to each other.

A CPN can be executed, i.e., there is a semantics specifying when a transition is *enabled* and what happens when the transition *occur*. A transition is enabled if certain tokens required for the action are present on the transition's input places. The input arcs to the transition describe which tokens are needed for the transition to be enabled. For example, for transition `Lookup customer information and credit information` to occur, a token of type `LOAN` must be present on the place `Customer observed`.

When a transition occurs, all tokens that are needed for its enabling are consumed and tokens are produced on all output places as described by the outgoing arcs. For example, when transition `Lookup customer information and credit information` occurs, a token of type `LOAN` is consumed from `Customer observed` and depending on whether the status of the `LOAN` token is set to `refusal` or `ongoing`, the token will be moved to either `Early refusal` or `Ready for advising`.

A CPN may consist of multiple modules, organized in a hierarchy. The RCPN consists of 7 modules and the module shown in Figure 3 is the top level model.

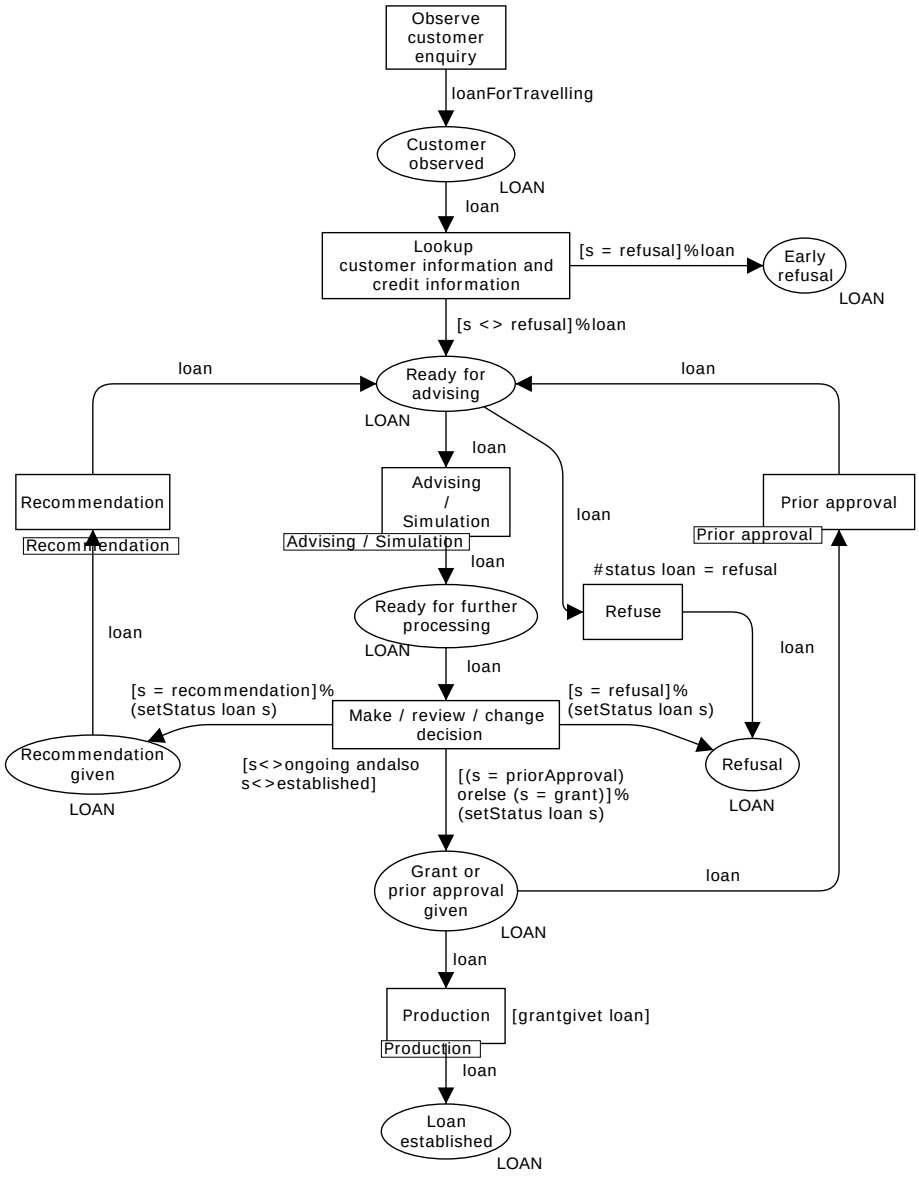

Fig. 3. Extract of the RCPN.

As placeholders for modules on lower levels, so-called substitution transitions are used. A substitution transition is represented as a rectangle with a small box with the module name near it. For example, `Advising / Simulation` is a substitution transition. This means that the details of how advising and what is called simulation is done are modeled on another module of the model. Note that

```
colset LOAN = record caseid: INT
                   * customer: STRING
                   * responsible: STRING
                   * status: STATUS
                   * amount: AMOUNT
                   * monthlyFee: MONTHLYFEE
                   * interestRate: INTEREST
                   * duration: DURATION
                   * purpose: STRING
                   * account: INT;
```

Fig. 4. LOAN type.

in the jargon used in the banks, *simulation* means that an adviser does some calculations and suggests various values for monthly payment, interest rate, and loan period to a customer.

The RCPN describes the control flow of actions which can happen from the point when a customer makes a blanc loan application until the request is either refused or established and in what sequence actions can occur. In the following, we describe how a single loan application is handled.

When transition `Observer customer enquiry` occurs, a `LOAN` token is put on `Customer observed`. Two things can happen at this point: (1) The adviser refuses the loan right away, e.g. if he knows that the customer has a bad credit history; (2) the adviser agrees to handle the loan application. In (1), the blanc loan application is terminated and a `LOAN` token is put on the place `Early refusal`. In (2), a `LOAN` token is put on `Ready for advising`. When the loan is in the state `Ready for advising`, it can go into the module `Advising / Simulation`. This is a module which models the creation of a task in a advisers task list and the first blanc loan application setup; i.e. how much money to loan, at which interest rate and so on. After this is done the `LOAN` token is put on `Ready for further processing`.

A choice is made at transition `Make / review / change decision`. This models the choice which the adviser must take at this point. The choices are *grant*, *recommendation*, *prior approval* or *refusal*, which are explained below.

- *Grant* is given if the blanc loan application is reasonable this choice is made. The `LOAN` token will be moved to `Grant or prior approval given` and then into the `Production` module. Here the blanc loan is finalized. This involves printing some necessary documents. Finally, the token is placed on the place `Loan established`.
- *Recommendation* means that the adviser's manager must make the decision. E.g., advisers are only permitted to grant loans up to a certain amount of money. This is modeled by the `LOAN` token being moved to `Recommendation given`. After this point it can enter the `Recommendation` module in which the behavior of the manager changing the status of the loan to either grant or refusal is modeled. After the module has executed, the token is moved

to `Ready for advising`. The behavior described earlier from this point can occur again.
- *Prior approval* is the case where the adviser accepts the blanc loan application but needs more information from the customer before continuing to process the blanc loan. Here the token is moved to `Grant or prior approval given`. From here it can enter the module `Prior approval` which models the situation where the processing of the loan is postponed. After this has been executed, the `LOAN` token is moved to `Ready for advising`. If a blanc loan has been given a prior approval it will always be granted at some point.
- *Refusal* applies if the adviser and customer cannot agree on a loan setup the application is refused. This is modeled by moving the `LOAN` token to `Refusal`. Then no other activities are possible for that particular loan application; this models that the case is ended.

All runs of a blanc loan application either end up in the state `Early refusal`, `Refusal`, or `Loan established`. The first two are for cases where applications are refused and the third is for cases where applications are approved.

4 From Requirements Model to Workflow Model

In this section, we address translation *T1* of Figure 1, which takes the RCPN described in the previous section and transforms it into a workflow model represented in terms of a CWN. We first motivate the need for a workflow model, then we introduce the class of CWNs, and finally, we present the CWN model for our case study.

4.1 Requirements Models Versus Workflow Models

In an RCPN, tokens, places and transitions may be used to represent arbitrary concepts relevant for requirements engineering. For example, tokens are used to represent customers, meeting rooms, conflicts, office equipment, paper document, etc. Some of these concepts have no counterpart in the final system. Given that a workflow management system is to be used, the vocabulary must be limited to the concepts supported by that system. For example, a workflow management system may know about cases, case attributes, tasks, roles, etc. but may not support concepts like meeting rooms and conflicts. Therefore, we propose a *system-independent language in-between the requirements level and the implementation level*. To do this, we first define some standard terminology and discuss differences between the requirements-level and workflow-level models.

Workflow processes, like the processing of blanc loans, are *case-driven*, i.e., tasks are executed for specific cases. A *case* may represent a blanc loan, but also the request to open a bank account, a customer complaint, or an insurance claim. These case-driven processes, also called *workflows*, are marked by at least the following three dimensions: (1) the *control-flow* dimension, (2) the *resource* dimension, and (3) the *case* dimension [1]. The control-flow dimension is concerned with the partial ordering of *tasks*. The tasks which need to be executed are

identified and the routing of cases along these tasks is determined. Conditional, sequential, parallel and iterative routing are typical structures specified in the control-flow dimension. Tasks are executed by *resources*. Resources are human (e.g., an adviser) and/or non-human (e.g., a printer). In the resource dimension these resources are classified by identifying *roles* (resource classes based on functional characteristics) and *organizational units* (groups, teams or departments). Each resource may have multiple roles and belong to multiple organizational units. For the execution of a task, a collection of resources may be required. (Although most workflow management systems assume only one resource to be involved in the execution of a task.) The required properties of the resources involved may be defined in terms of roles and organizational units, e.g., task "Recommend" needs to be executed by a "manager" of the "loans department". Both the control-flow dimension and the resource dimension are generic, i.e., they are not tailored towards a specific case. The third dimension of a workflow is concerned with individual cases which are executed according to the process definition (first dimension) by the proper resources (second dimension). Cases may have attributes, e.g., some account number or the interest rate. The attribute names and their types are generic while the concrete attribute values are specific for a concrete case.

In summary, workflow models should have the following three properties:

W1 The workflow model should use a *restricted vocabulary* common for workflow management systems as indicated above. Only concepts such as case, task, resource, role, organizational unit, and attribute may be used to construct workflow models.

W2 The workflow model includes *only actions* (i.e., tasks) which are to be *supported by the workflow management system* (i.e., it includes no tasks, e.g. manual actions, the system will not be aware of).

W3 The workflow model *refines* selected parts of the requirements model to enable system support. Note that granularity of tasks may not be dealt with in detail at the requirements-level. However, at the workflow-level, the splitting and/or merging tasks is important as it directly influences the implementation of the system.

This paper proposes CWNs as a workflow modeling language sitting in-between the RCPN and BPEL, as shown in Figure 1.

4.2 Colored Workflow Nets

A *Colored Workflow Net* (CWN) is a CPN as defined in [16, 20]. However, it is restricted as indicated in Section 4.1. Note that a CWN covers the *control-flow* perspective, the *resource* perspective, and the *data/case* perspective, and abstracts from implementation details and language/application specific issues. A CWN should be a CPN with only places of type `Case`, `Resource` or `CxR`. These types are as defined in Figure 5.

A token in a place of type `Case` refers to a case and some or all of its attributes. Tokens in a place of type `Resource` represent resources. Places of type

```
colset CaseID =union C:INT;
colset AttName = string;
colset AttValue = string;
colset Attribute = product AttName * AttValue;
colset Attributes = list Attribute;
colset Case = product CaseID * Attributes timed;
colset ResourceID = union R:INT;
colset Role = string;
colset Roles = list Role;
colset OrgUnit = string;
colset OrgUnits = list OrgUnit;
colset Resource = product ResourceID * Roles * OrgUnits timed;
colset CxR = product Case * Resource timed;
```

Fig. 5. Places in a CWN need to be of type `Case`, `Resource` or `CxR`.

`CxR` hold tokens that refer to both a case and a resource. Such places are used if resources need to execute a sequence of tasks for the same case, e.g., chained execution.

A CWN where all places of type `Resource` are removed should correspond to a *Sound Workflow Net* (sound WF-net) as defined in [1]. Although WF-nets have been defined for classical Petri nets it is easy to generalize the definition to CPN as discussed in [1, 2]. The basic requirement is that there is one source place and one sink place and all other nodes (places and transitions) are on a path from source to sink. Moreover, given a token on the input place, eventually one token should appear on the output place and the rest of the places should be empty.

There should be *conservation of cases and resources*. This can be formulated in terms of colored place invariants [16]. For every resource place r there should be a (colored) place invariant associating weight 1 to r and selected places of type `CxR` (if any) and weight 0 to all other places. Moreover, there is a (colored) place invariant associating weight 1 to the source and sink place and positive weights to all other places of type `Case` or `CxR` (and weight 0 to all places of type `Resource`).

Transitions correspond to tasks supported by the workflow system. They should not violate the soundness and conservation properties mentioned above. Therefore, we propose the following *guidelines*.

- The following variables should be defined: `c`, `c1`, `c2`, `c3`, etc. of type `Case` and `r`, `r1`, `r2`, `r3`, etc. of type `Resource`. Using a fixed set of variables facilitates both the interpretation and automated translation of CWNs. For similar reasons we structure the arc inscriptions and guards (see below).
- Arcs from a place to a transition (i.e., input arcs) should only use the following inscriptions:
  - `c`, `c1`, `c2`, `c3`, etc. for arcs from places of type `Case`.
  - `r`, `r1`, `r2`, `r3`, etc. for arcs from places of type `Resource`.

  - `(c,r)`, `(c,r1)`, `(c1,r1)`, `(c2,r1)`, etc. for arcs from places of type `CxR`.
- Arcs from a transition to a place (i.e., output arcs) should satisfy the same requirements unless they are conditional and of the form `[C]%c` or `[C]%(c,r)` (where `C` is some condition depending on the case attributes).
- Guards should be logical expressions created only using functions such as `match`, `has_role`, `has_orgunit`, etc. Function `match` can be used to make sure that all arc inscriptions involved in one transition bind to the same case id. Function `has_role` can be used to make sure that the resource selected has a given role. Function `has_orgunit` can be used to make sure that the resource selected is a member of a specific organizational unit. The definitions of these functions are straightforward but beyond the scope of this paper.
- Each transition should have a code region. This is executed when the transition occurs. In the context of CWNs, code regions are used to link the case attributes of the input tokens to the output tokens. Since arc expressions cannot manipulate the case content, i.e., only route cases, this is the only way of changing the attributes of a case.

As indicated above, guards should only be used to make sure that only tokens corresponding to the same case are merged and that the tasks are executed by the proper resources. A typical guard is: `match(c1,c2) andalso has_role(r1,"manager") andalso has_orgunit(r1,"loans")`. We do not put any requirements on the code regions other than that the conservation of cases and resources should be guaranteed. Note that tasks are executed by humans using applications. In a CWN model one can try to model these but it will always be an approximation. Typically it is impossible to make a complete/accurate model of some bank application or bank advisor. Therefore, we can only try to approximate them and mapping the CPN code regions to some implementation language (e.g., BPEL) will have to be partly manual either at implementation/configuration-time or at run-time.

In this paper, we will not give a formal definition of CWNs. Rather, we focus on the CWN for the AP system.

4.3 Workflow Model for AP

Figure 6 shows the CWN, which is derived from the RCPN of Figure 3. The CWN reflects the parts in the RCPN that should be orchestrated by the workflow system. As stated in W2, it should not contain actions which are not supported by the system. We therefore leave out transitions `Observer customer enquiry`, `Lookup customer information and credit information` and `Refuse`, and the places `Customer observed`, `Early refusal`, and `Ready for advising`. The rest of the RCPN has been mapped into the CWN form as shown in Figure 6.

It can be observed that the CPN shown in Figure 6 is a CWN as defined in Section 4.2. It is easy to see that all places are of type `Resource`, `Case` or `CxR`. If the resource places are removed from the model, a WF-net emerges, i.e. all nodes are on a path from start to end node. In this context it is also evident that the net has a start and end node and is indeed sound, i.e., all cases begin

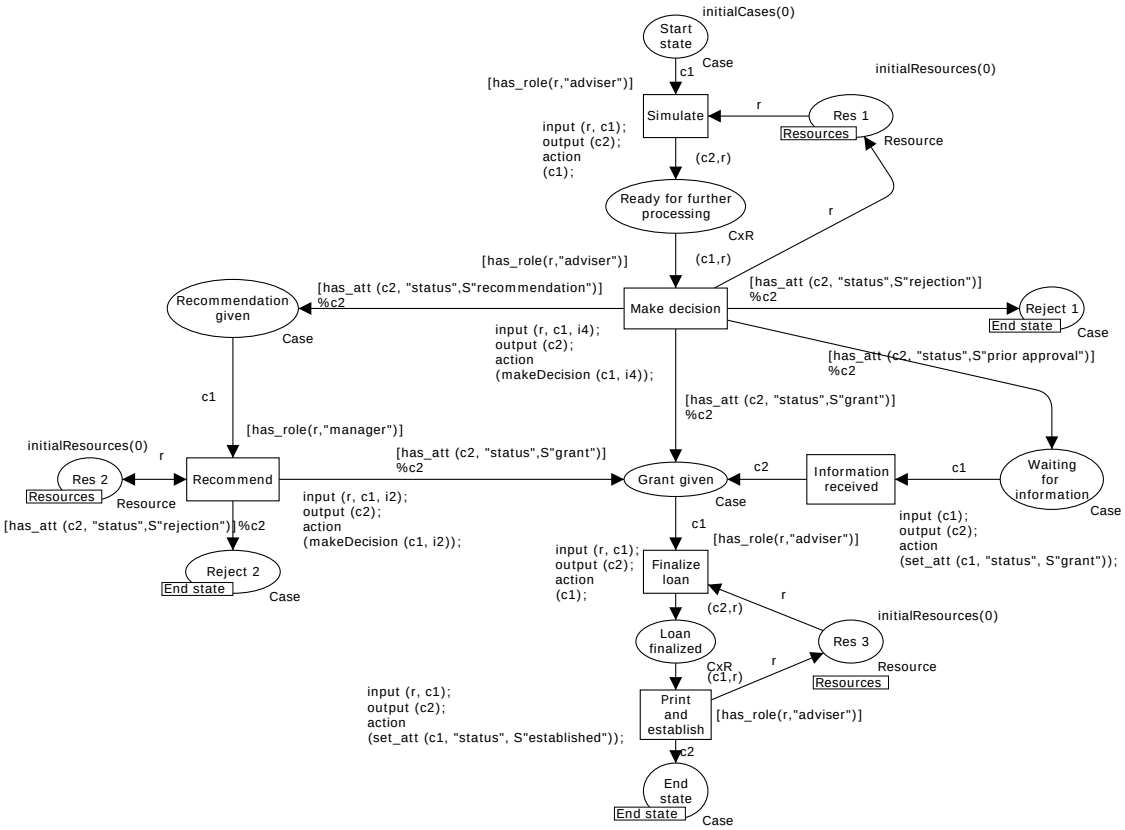

Fig. 6. The result of translation *T2*: the CWN for AP.

from the place `Start` and end up in one of the places in the `End state` fusion set. Also the guidelines regarding the arc inscriptions and guards of transitions representing tasks are satisfied. Note the code regions in Figure 6 linking `input` to `output` via an `action` part.

Apart from satisfying properties (W1)-(W3) mentioned in Section 4.1, as a natural result of being created later than the requirements model, the workflow model improves a number of things — the workflow model corrects some errors and shortcomings in the RCPN and improves the modelling of a number of aspects. As an example, the RCPN contained traces which are not possible in the real world. For example, a loan could be given a prior approval and then rejected. In the real world a loan can only be granted if a prior approval had been given. Therefore we have restricted the behavior in the workflow model to avoid such execution paths.

In the following we give a structured translation of the requirements model to the workflow model beginning from the first activity in a blanc loan application process.

Generally, resource places are put in the model to reflect which human resources are available/needed at certain steps in the process. Note that place fusion [16] is used in Figure 6, i.e., the three resource places are actually the same place.

The first activity that occurs in the process is that an adviser proposes a loan setup in the module `Advising / Simulation`, this is mapped to transition `Simulate` which is the first possible activity which can occur in the workflow model. (Recall that the term "simulation" is part of the jargon used in banks and refers to a specific activity.) For this to happen an adviser must perform the work which is reflected by the arc from `Res 1` and the guard `[has_role(r,"adviser")]` on `Simulate`. The adviser resource is not put back before a decision about the loan is made in the transition `Make decision`. It is modeled in this way to reflect that it is the same adviser that proposed the loan which makes the decision.

The decision can have four possible outcomes as in the RCPN: (1) grant, (2) recommendation, (3) prior approval or (4) refusal. In contrast to the RCPN it is not possible to make this decision twice, i.e., the flow of the loan case will not loop back to the decision node `Make/review/change decision`. Below we describe how each of the four following scenarios are translated:

- In the RCPN, the case went into the module `Production` if a grant was given. This is reflected by the transitions `Finalize loan` and `Print and establish`. These are both done by the same adviser. This is reflected by the arcs connected to resource place `Res 3` and the guard of `Simulate`.
- If recommendation is given, the activity `Recommend` can occur. This is the translation of the module `Recommendation` in the requirement CPN. It is modeled by two outgoing arcs from the transition `Recommend` and a required manager resource from the resource place `Res 2`. In case of a refusal the case ends which is reflected by the case being moved to the end state of the workflow model.
- If a prior approval is given, the case moves to a waiting position on the place `Waiting for information` until the information arrives. When this happens, `Information received` occurs and the status of the loan is set to grant and the case follows the path for a loan with a grant from this point.
- When a refusal is issued, the blanc loan is moved to the end state with a rejection as status to reflect the termination of the case.

5 From Workflow Model to Implementation

Now we focus on translation *T2* of Figure 1. We do not present our technique to map a CWN onto BPEL in detail. For more details we refer to a technical report defining the mapping from WF-nets to BPEL [4].

The key issue is that CWNs can be used to semi-automatically generate BPEL code. It is possible to fully automatically generate template code. However, to come to a full implementation, programmers must manually complete the work, e.g., by providing the "glue" to existing applications and data structures.

BPEL offers many routing constructs. The atomic activity types are: `invoke` to invoke an operation of a web service described in WSDL, `receive` to wait for a message from an external source, `reply` to reply to an external source, `wait` to remain idle for some time, `assign` to copy data from one data container to another, `throw` to indicate errors in the execution, `terminate` to terminate the entire service instance, and `empty` to do nothing. To compose these atomic activity types, the following structured activities are provided: `sequence` for defining an execution order, `switch` for conditional routing, `while` for looping, `pick` for race conditions based on timing or external triggers, `flow` for parallel routing, and `scope` for grouping activities to be treated by the same fault-handler. Typically there are multiple ways of realizing the same behavior, e.g., the `flow` construct can be used to build sequences and switches but also the `sequence` and `switch` constructs may be used.

We have developed an iterative approach to "discover" patterns in CWNs that correspond to constructs in BPEL and generate template code based on this. The approach works as follows. First, the CWN is mapped onto an annotated WF-net. (Figure 7 shows an example of such mapping.) This involves removing the resource places and resource-related inscriptions, removing the color-related information, and replace transitions that represent choices by a small network. The resulting "uncolored" WF-net is annotated with information that can be used for the BPEL translation, e.g., conditions for choices. In the annotated WF-net we try to select a *maximal sequence component*, i.e., a sound workflow subnet that is both a state machine and a marked graph [1, 26]. This part of the net is reduced to a single transition representing the sequence (cf. Figure 7). If there is no maximal sequence component, the algorithm looks for some other structured component (e.g., a "switch", "pick", "while", or "flow") that is maximal in some sense. Again this component is reduced to a single transition. (The approach also allows for ad-hoc and reusable components with a corresponding BPEL mapping.) By applying these rules iteratively the Petri net is parsed completely and the parse tree is used to generate the BPEL code as illustrated by Figure 7.

We have applied the algorithm to map the CWN shown in Figure 6 onto BPEL code and tested this using IBM WebSphere Studio. The BPEL code and screenshots of WebSphere Studio can be downloaded from `http://www.daimi.au.dk/~krell/CoopIS05/`.

6 Related Work

Since the early nineties, workflow technology has matured [14] and several textbooks have been published, e.g., [2, 10]. Petri nets have been used for the mod-

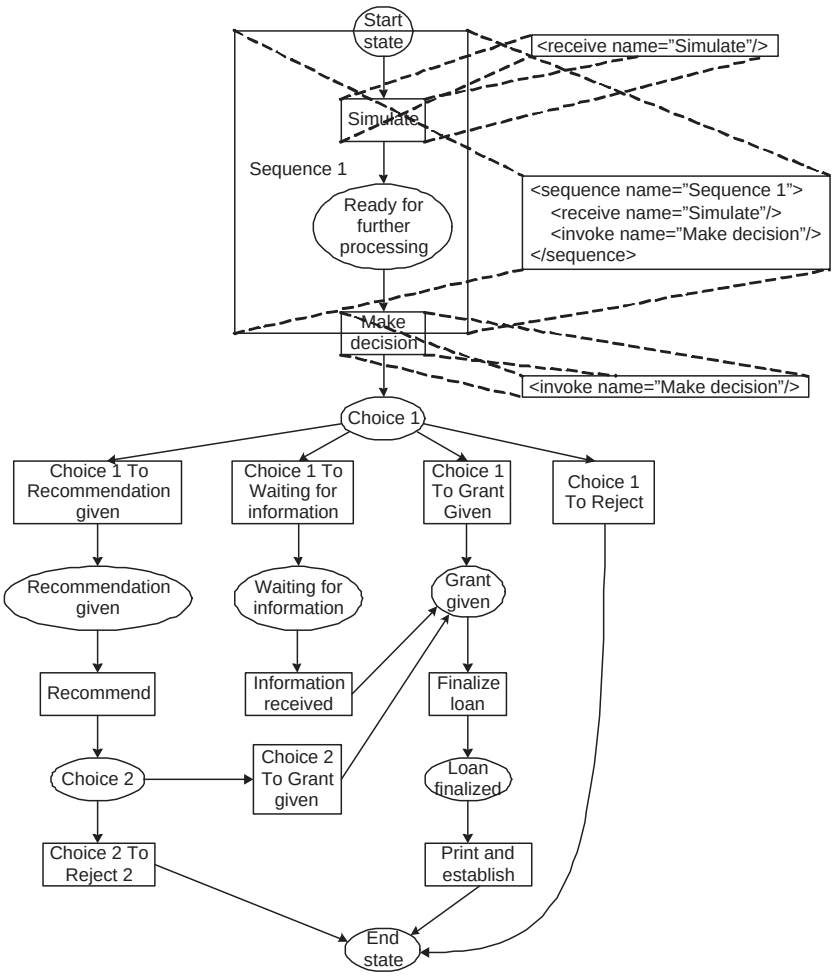

Fig. 7. The "uncolored" WF-net with some snippets of generated BPEL code.

eling of workflows [2, 7, 10] but also the orchestration of web services [22]. The Workflow Management Coalition (WfMC) has tried to standardize workflow languages since 1994 but failed to do so. XPDL, the language proposed by the WfMC, has semantic problems [1] and is rarely used. In a way BPEL [5] succeeded in doing what the WfMC was aiming at. However, BPEL is really at the implementation level rather than the workflow modeling level or the requirements level (thus providing the motivation for this paper).

Several attempts have been made to capture the behavior of BPEL [5] in some formal way. Some advocate the use of finite state machines [13], others process algebras [12], and yet others abstract state machines [11] or Petri nets

[23, 21, 24, 25]. (See [23] for a more detailed literature review.) For a detailed analysis of BPEL based on the workflow patterns [3] we refer to [28].

Most papers on BPEL focus on the technical aspects. This paper focuses on the life-cycle of getting from informal requirements to a concrete BPEL implementation based on a concrete case study. Therefore, the work is perhaps most related to the work of Juliane Dehnert who investigated the step from informal business models expressed in terms of Event-driven Process Chains (EPCs) to workflow models expressed in terms of WF-nets [8, 9]. However, her work is mainly at theoretical level and does not include concrete case studies or mappings onto some implementation language.

The work reported in this paper is also related to the various tools and mappings used to generate BPEL code being developed in industry. Tools such as the IBM WebSphere Choreographer and the Oracle BPEL Process Manager offer a graphical notation for BPEL. However, this notation directly reflects the code and there is no intelligent mapping as shown in this paper. This implies that users have to think in terms of BPEL constructs (e.g., blocks, syntactical restrictions on links, etc.). More related is the work of Steven White that discusses the mapping of BPMN onto BPEL [27] and the work by Jana Koehler and Rainer Hauser on removing loops in the context of BPEL [19]. Our work differs from these publications in the following way: we address the whole life-cycle (i.e., not a specific step or a specific problem as in [19]), we provide algorithms to support the mapping (unlike, e.g., [27]), and we use CPNs as a basis (i.e., start from a language with formal semantics).

The translation from WF-nets to BPEL (translation *T2*) is described in more detail in a technical report [4]

7 Conclusions

In this paper, we have used a real-life example (the new AP system of Bankdata) to go from a requirements model to a proof-of-concept BPEL implementation using the CPN language and CPN Tools. Figure 1 summarizes our approach and shows the different models and translations proposed. The focus of this paper has been on translations *T1* and *T2*. Essential for this approach is the use of the CPN language, first in unrestricted form (the RCPN) and then in restricted form (the CWN). The restrictions facilitate the automatic generation of (template) code.

In this paper, we introduced the CWN model and the translation to BPEL code. We believe that our approach can be generalized to other systems within and outside Bankdata. We also believe that our approach can be modified for other target implementation languages (e.g., languages used by systems of TIBCO/Staffware, FLOWer, COSA, and FileNet). Further case studies are needed to prove this point. We also aim at concrete tool support for the translation of CWN to BPEL. At this point in time, we provide only a manual procedure to accomplish this. We plan to develop a dedicated tool for this.

Another direction for future research is to develop techniques, tools, and animations specific for CWN. The CWN model can be seen as the high-level variant of the classical workflow nets, adding the data and resource perspectives. For workflow nets there are strong theoretical results, dedicated editors, and analysis tools [1, 26]. The goal is to offer similar support for CWNs. For example, it is interesting to explore different notions of soundness including the data and resource perspectives [1, 8, 15].

Acknowledgements We thank Bankdata for allowing us to participate in the AP project. We thank the users and analysts we have worked with, in particular Gert Schmidt Sofussen, who has contributed significantly to the RCPN.

References

1. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.
2. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
3. 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.
4. W.M.P. van der Aalst and K.B. Lassen. Translating Workflow Nets to BPEL4WS. BETA Working Paper Series, Eindhoven University of Technology, Eindhoven, 2005.
5. T. Andrews, F. Curbera, et al. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
6. C. Bossen and J.B. Jørgensen. Context-descriptive Prototypes and Their Application to Medicine Administration. In *Proc. of Designing Interactive Systems DIS 2004*, pages 297–306, Cambridge, Massachusetts, 2004. ACM.
7. P. Chrzastowski-Wachtel. A Top-down Petri Net Based Approach for Dynamic Workflow Modeling. In *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 336–353. Springer-Verlag, Berlin, 2003.
8. J. Dehnert. *A Methodology for Workflow Modeling: From Business Process Modeling Towards Sound Workflow Specification*. PhD thesis, TU Berlin, Berlin, Germany, 2003.
9. J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
10. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems*. Wiley & Sons, 2005.
11. D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, March 2005.
12. A. Ferrara. Web services: A process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.

13. J.A. Fisteus, L.S. Fernández, and C.D. Kloos. Formal verification of BPEL4WS business collaborations. In *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, August 2004. Springer-Verlag, Berlin.
14. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
15. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.
16. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
17. J.B. Jørgensen and C. Bossen. Executable Use Cases: Requirements for a Pervasive Health Care System. *IEEE Software*, 21(2):34–41, 2004.
18. J.B. Jørgensen and K.B. Lassen. Aligning Work Processes and the Adviser Portal Bank System. In *International Workshop on Requirements Engineering for Business Need and IT Alignment*, 2005.
19. J. Koehler and R. Hauser. Untangling Unstructured Cyclic Flows A Solution Based on Continuations. In *CoopIS 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 121–138, 2004.
20. 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.
21. A. Martens. Analyzing Web Service Based Business Processes. In *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
22. M. Mecella, F. Parisi-Presicce, and B. Pernici. Modeling E-service Orchestration through Petri Nets. In *Proceedings of the Third International Workshop on Technologies for E-Services*, volume 2644 of *Lecture Notes in Computer Science*, pages 38–47. Springer-Verlag, Berlin, 2002.
23. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-13, BPMcenter.org, 2005.
24. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
25. H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL Processes using Petri Nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78. Florida International University, Miami, Florida, USA, 2005.
26. 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.
27. S. White. Using BPMN to Model a BPEL Process. *BPTrends*, 3(3):1–18, 2005.
28. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.