Foundations of Aspect Oriented Business Process Management

Amin Jalali and Petia Wohed

Department of Computer and Systems Sciences, Stockholm University, Sweden
{amin-jal,petia}@dsv.su.se

Abstract. Reducing complexity in Information Systems is a main concern in both research and industry. One strategy for reducing complexity is separation of concerns. This strategy advocates separating various concerns, like security and privacy, from the main concern. It results in less complex, easily maintainable, and more reusable Information Systems. Separation of concerns is addressed through the Aspect Oriented paradigm. This paradigm has been well researched and implemented in programming, where languages such as AspectJ have been developed. However, the research on aspect orientation for Business Process Management is still at its beginning. While some efforts have been made proposing Aspect Oriented Business Process Modelling, it has not yet been investigated how such modelling should be enacted in a Workflow Management System. In this paper, we define a set of requirements that specifies how aspect oriented business process models should be enacted in a Workflow Management System. Furthermore, we design a Coloured Petri Nets model for a service that fulfils these requirements. This service extends the behaviour of a Workflow Management System with support for execution of aspect oriented business process models. The model is verified and validated using state space analysis and scenarios.

Keywords: Business Process Management, Workflow Management Systems, Aspect Oriented, Coloured Petri Nets, Weaving

1 Introduction

Reducing the complexity of models is an important issue in the Business Process Management (BPM) area. Business process models tend to quickly become complex [3], which makes them difficult to communicate, use, maintain and validate [20]. Various approaches have been proposed to reduce the complexity of process models (e.g. [15, 22, 23]). Some of these approaches have been analysed and systemised as a collection of patterns [20]. One of the patterns is called orthogonal modularization, and its purpose is to reduce the complexity of a model by separating different aspects of a process, such as security and privacy. Traditionally, these aspects are defined in a single process model, hence adding to the complexity of the model [24]. In contrast, Orthogonal modularization models the aspects as separate processes in individual models. These processes are related to the main process, where they represent different pieces of the puzzle. The business process is described when all pieces of the puzzle are put together. The
mechanism that puts all aspects and the main process model together is called weaving, while the whole technique is called Aspect Oriented Modularization.

Aspect oriented modularization has been defined on the modelling level in the BPM area (e.g. [12, 14, 9]). However, there are still no results in the literature on how to address enacting and analyzing aspect oriented modularization in BPM. Thus, the aspect oriented principle can be interpreted in different ways by different people. Therefore, a rigorous and general model is required to formally specify how the enactment of aspect oriented modularization should be done in a Workflow Management System (WFMS).

In this paper, we propose such a model. This model is defined using Coloured Petri Nets (CPN). Petri nets are good candidates for modelling WFMSs, because they meet all the functional requirements [2]. Thus, the designed model is formal and could be used as a blueprint for a service, called Aspect Service, for extending a WFMS with support for aspect oriented modularization. Moreover, Petri nets models support analysis of the models through an extensive number of analysis techniques [2]. We will prove the soundness of our model using state-space analysis.

The reminder of this paper is structured as follows: Section 2 presents a background of aspect oriented business process modelling. Section 3 describes the architecture of the Aspect Service and defines the requirements for the service. Section 4 describes the formalization of the service, and Section 5 presents the verification and the validation of the solution. Finally, Section 6 discusses related work, and Section 7 concludes the paper and outlines directions for future work.

2 Background

Process models encompass different activities which address different concerns of business processes. Concerns are non-functional requirements of a business process which are a matter of interest for stakeholders. Charfi enumerates compliance, auditing, business monitoring, accounting, billing, authorization, privacy and separation of duties as examples of concerns [12]. Some of these concerns are not limited to a single business process but can be repeated in several business processes.

For example, it is compulsory in Swedish public organizations to inform citizens if a decision is made on their applications. Therefore, an inform activity is scattered across all business processes that contain a decide activity. Moreover, a process may contain several decide activities, implying the need for several inform activities. If the inform activity is changed, or if the policy regarding the informing concern is modified, we have to find and update all business processes containing any decide activity. To be conformed to the law, when designing a new business process we should remember to add the inform activity after each decide activity. These efforts add costs in designing, updating and monitoring business processes, and increase the risk of inconsistency when updating processes due to changes in a concern. Moreover, concerns could not be reused since they are implemented separately in each business process. Therefore, business processes become more complex, less reusable and more costly to design and maintain.
The Aspect Oriented Paradigm addresses these problems by separating different concerns from the main process. There are various works (e.g. [12, 14, 9]) which provide means for aspect oriented business process modelling. Aspect Oriented Business Process Modeling Notation (AO4BPMN) [12] is one such approach that defines the terminology and suggests a notation for modelling processes according to the aspect oriented principle. We describe the AO4BPMN using an example. Figure 1 describes a simplified version of the Transfer Money Process in the banking domain. First, a customer fills information. Then, if she is transferring money to her own account, the transfer is performed. Otherwise, she must sign the transaction beforehand. Finally, the transaction is archived. The Sign Transaction activity is part of the security aspect, and the Archive Information is part of the logging aspect. These aspects describe different concerns related to the Transfer activity.

AO4BPMN supports modelling the above process as Figure 2. The concerns are removed from the main process and modelled separately through aspects. Hence, the Transfer Money Process will contain the Fill Information and Transfer activities. An aspect is the realisation of a concern through one or a number of processes. Each aspect is modelled in a separate model, i.e. the Logging Aspect and the Security Aspect. Each aspect model is annotated with an Aspect label. An aspect may have one or several advices, each anno-
ated with an Advice label. An advice captures a specific part of a concern that should be considered if a condition is fulfilled. This condition is called Pointcut. The pointcut shows when and where the advice should be integrated with the process model. The possible points of integrations are called Join Points. In AO4BPMN, these are activities. For instance, Fill Information and Transfer activities are both examples of join points. A join point can be related to an aspect by defining a pointcut. In such case, a join point is called advised join point. In the example, the advice in the Logging Aspect is related to a pointcut, named Archive. This pointcut is related to the Transfer activity through an annotation, which shows that the Logging Aspect is relevant to the Transfer activity. The Transfer activity is an advised join point.

An advice typically contains a PROCEED activity. This activity acts a “placeholder” indicating when the advised join point activity should be performed. In Figure 2 the Archive Information will be completed after the Transfer activity. There is also another pointcut which is related to the Transfer activity. It shows that the Transfer activity should be signed when a customer wants to transfer money to an account owned by someone else. This concern is modelled through the Security Aspect. This aspect contains Sign Transaction activity, which is performed before the Transfer activity.

AO4BPMN modelling increases reusability because an aspect can be related to different activities and even different processes. It also facilitates the maintenance of processes, because if a concern changes, the change would only be reflected in one place. Finally, it reduces the complexity of a process model as it reduces the number of activities inside the process.

3 Overview of the Solution

In this section, we describe the foundations of a service, called Aspect Service, which complements WfMSs with support of aspect oriented modularization.

We define this service as a sub-service of the Worklet Service [6]. The Worklet Service is defined to support flexibility for business processes [5]. It is utilized because: (i) it provides a foundation for extending the behaviour of business process execution in a WfMS; (ii) it is based on Service Oriented Architecture (SOA), hence applicable for any WfMS; (iii) it is open-source and currently proven as an implementation for YAWL; and (iv) it is formalized through CPN, which enables reuse of relevant sub-nets [8].

For explaining the Aspect Service, we use a general example (see Figure 3A). The example contains a main process with four aspects, which are defined for one of its activities, activity B. The enactment of the business processes is managed through a WfMS. It results in five different process instances (see Figure 3Bii). The Aspect Service takes care of the weaving of the aspects to the main process. In this particular example, the execution sequence of the activities will be followed according to the flow which is illustrated in Figure 3Cii. Figure 3B shows the relationship between a WfMS, the Worklet Service and the Aspect Service.

The Aspect Service gets enabled upon receiving a constraint. In [7], four types of constraints are defined, i.e., CasePreConstraint, ItemPreConstraint,
Fig. 3. Interactions between the Aspect Service and a WfMS

*ItemPostConstraint* and *CasePostConstraint*. *ItemPreConstraint* and *ItemPostConstraint* represent the beginning and ending of a work item while *CasePreConstraint* and *CasePostConstraint* represent the beginning and ending of a case. Each time a work item gets enabled, a constraint of the type *ItemPreConstraint* is raised by the WfMS. The Aspect Service receives this constraint and performs two checks: a check on whether the workitem has a pointcut associated to it and if so, if the pointcut is met (recall that a pointcut is a condition). If the workitem is not related to a pointcut, or if a pointcut condition is not met, the execution of the workitem is proceed as usual. Otherwise, the Aspect Service starts the weaving of the corresponding aspect(s). We call this the *Initiating step*. This step represents the initiating requirements of the service. I.e. the service shall check if: a received workitem constraint is related to a pointcut; if a received workitem
constraint is related to a Proceed placeholder; if a received case constraint is related to the end of advice.

In fact, the same checks are also performed when an ItemPostConstraint is raised (which occurs at a work item completion). This is a way to indicate the beginning and the end of an advised join point and to distinguish it from normal workitems. Graphically, this is shown with two dots on the task symbol.

The weaving is a process that orchestrates the enactment of the main business process module and its related aspects. Both the main process and related aspects are enacted in the WfMS as separate business processes. Therefore, the weaving is performed by the Aspect Service through receiving and sending sets of messages from/to the WfMS. This orchestration is performed in four steps representing four requirements, which are described below.

1. **Launching**: When the Aspect Service is activated, it sends a message to the WfMS to suspend the main process. When the suspension is confirmed by the WfMS, the Aspect Service sends messages to the WfMS for launching the relevant advices. Hence, the initiating requirement is defined as the Aspect Service shall support the WfMS to suspend the execution of an advised join point (R1.1) and launch its related aspects (R1.2).

2. **Pausing**: When all advices reach the Proceed placeholder, the Aspect Service sends messages to the WfMS to suspend the advices. However, advices that do not have a Proceed placeholder will reach to their End. Once the relevant advices have been suspended or ended, the Aspect Service orders unsuspension of the advised join point. We call an advice which contains the Proceed placeholder an explicit advice and an advice which does not have any Proceed placeholder an implicit advice. Hence, the pausing requirement is defined as the Aspect Service shall support the WfMS to suspend launched aspects upon reaching the Proceed placeholder (R2.1) and unsuspend the advised join point (R2.2).

3. **Resuming**: After the advised join point has been completed, the WfMS raises an ItemPostconstraint (see the filled dot in Figure 3Ci). The Aspect Service sends messages for suspension of the advised join point and unsuspension of the corresponding advices. Then, it sends messages to force complete the Proceed placeholders, so the aspects can be continued. Hence, the resuming requirement is defined as the Aspect Service shall support the WfMS to suspend the execution of an advised join point (R3.1) and unsuspend the execution of its paused aspects (R3.2).

4. **Finalizing**: When all advices are ended (i.e. their CasePostconstraints have been raised), the Aspect Service sends message to the WfMS to unsuspend the advised join point. Hence, the weaving is completed, and the control of the main process is handed back to the WfMS. Hence, the finalizing requirement is defined as the Aspect Service shall support the WfMS to unsuspend the advised join point when its related aspects are finished (R4).

During all this steps, the data is synchronised between the main process and its aspects. In the next section, we describe the CPN model for the Aspect Service.
4 Formalization

The formalisation of the Aspect Service is designed through a hierarchical Coloured Petri Nets. The solution is a three level model. The top-level module captures the behaviour of the Initiation of the service (see Figure 4). The second level captures the weaving behaviour (see Figure 5). This model contains four modules for capturing the requirements related to steps 1 to 4, which were described in the previous section. It also contains a module for communicating with the WfMS and persisting data, which is needed for the weaving of the aspects to the main process. These five modules build the third level of the CPN Model.

The model contains 57 colour sets and 33 functions. The complete model with detailed definition of colour sets, variables and functions can be downloaded from [1]. We re-used some of the colour sets, variables and functions from the Worklet CPN models [8]. In the following subsections, we explain parts of the solution.

4.1 Level 1: the top-level module

The interaction of the Aspect Service and a WfMS is realized through passing a number of messages. These messages are named constraints and commands. Constraints are the messages raised by the WfMS, and Commands are the messages invoked by the Aspect Service. All these messages should be supported by the WfMS. We used YAWL [13] as an example system and the name of the messages as defined in YAWL. However, it should be noted that although the names may differ from one system to another, the messages are generic. The constraint messages are WorkitemConstraint and CaseConstraint, as depicted in Figure 3B. The commands messages will be explained later. The raising of one of these messages is signified in the CPN model in Figure 4 as a token arriving in the workitemConstraint or caseConstraint places correspondingly. We have highlighted this by shading these places in the model. In another words, these places are the starting points of the net.

The first level of the net addresses the initiating requirement by processing the received constraints. The constraints are related to workitems or cases. Workitem constraints represent the enabling and ending of a workitem. A workitem is an instance of a task. Each task can be associated with a pointcut, except the Proceed placeholders. Therefore, these workitems are checked if they meet the conditions of their pointcuts. The net perform this check using the matchPointcut transition. If the pointcut is met, the bold part of the net is executed. The result of this execution is a token in AdvisedJP or AspectInfo places. If the constraint is an ItemPreConstraint, a token is produced in AspectInfo place; otherwise if the constraint is an ItemPostConstraint, a token is produced in AdvisedJP place. However, if the token in the workitemConstraint place represents a constraint for a Proceed placeholder, it will be consumed by the isProceedCmd transition. This transition produces a token in Proceed place as a result.

In order to fulfil the initiating requirement, the case constraints should also be addressed. Therefore, the endAdvice transition checks the tokens in the
caseConstraint place. This transition produces a token in completeAdvice place if the case constraint is related to an advice.

In conclusion, this net controls how the weaveAspect net is enabled. The weaveAspect net is responsible for addressing the weaving requirements and is described in the next section.

4.2 Level 2: the Weaving and selectPointcut modules

There are two sub-nets at the second level, i.e. the selectPointcut and the weaveAspect modules. The selectPointcut module in the Aspect Service net is reused from the Worklet Service [8]. We use this net to check whether a pointcut is met.

The weaveAspect sub-net (see Figure 5) contains five module, which address the weaving requirements. The net contains Launching, Pausing, Resuming and Finalizing sub-nets which handle the four weaving steps, depicted in Figure 3Ci. There is also a Core module which is responsible for persisting the data among these four steps. Furthermore, the Core module is responsible for sending messages to the WfMS. All these five sub-nets has a common place, ICore, which is both input and output place for all of them.

The tokens in this place symbolize different messages passed through the nets. The color set of the ICore place is COREMSG. COREMSG is the product of a command name and a parameter which is a list (see Figure 6). The bold commands, in the CMD colour set, represent the commands which are shown in Figure 3B. Each of the nets produces and consumes specific sets of tokens with different commands. The ICore place is the foundation for the communication between the nets. A brief description of some of these nets is given below.

1 In [8] this net is called ‘CPN: Evaluating the Rule Tree’.
4.3 Level 3: the leaves modules

The **Launching** sub-net, depicted in Figure 7, is designed to address the launching requirements. The **launchAspect** transition produces four tokens in the **ICore** place. One of these tokens, with **suspendWorkItem** command, is used by the **Core** module to send a message to the WfMS to suspend the advised join point (fulfilling R1.1). One of the tokens, with **initAdvice** command, is consumed by the **Core** module to persist the number of advices related to the advised join point. The other two tokens, both containing **getAdviceNumber** command, are used to retrieve the number of advices for the advised join point. The **launchAspect** transition also produces a token in **Aspect** place. The **getnextAdvice** transition extracts all advices of the pointcut, and produces individual tokens for them in **advice** place. After all advices are extracted, the advised join point is suspended, and the number of advices related to the advised join point are retrieved, the **enableLauncher** transition will produce some tokens in **AdvisedJP** place. The number of tokens is equal to the number of advices, so **initAdvice** transition fires once per advice.

As mentioned earlier, each advice is enacted as a separate business process. The Aspect Service does not know the ids of the advices before launching but needs to keep track of them once they become known. To distinguish the id for each advice, the Aspect Service must launch them individually. This is captured by the bold parts in the net where place **ExclFlag**, containing most one token at the time, enforces a sequential launching.

Basically, the launching is realised in three steps. First the input parameters are retrieved, second an advice is launched, and third the relation between the advised join point and the launched advice is stored. The **first step** is realized through the **initAdvice** transition. This transition produces three tokens. The first token, in place **ICore**, is used by the **Core** sub-net to send a message to the WfMS to retrieve the parameters of the advice. The second token, in **joinPointID** place, keeps the advised join point id. This information is later used to persist the relation with the advice. The third token, in **AdviceInfo**
colset PARAM = product STRING * STRING;
colset PARAMS = list PARAM;

colset CMD = with none 
| initAdvice | setAdviceInfo 
| getAdviceInfoByCase | rspGetAdviceInfoByCase 
| getAdviceInfoByItemID | rspGetAdviceInfoByItemID 
| getAdviceNumber | rspGetAdviceNumber 
| saveJoinpointID | getJoinpointID | rspGetJoinpointID 
| removeAdvice 
| noMoreAdvice 
| parentIDFlag | resumeFlag 
| getInputParams | rspGetInputParams 
| unsuspendWorkItem | ackUnsuspendWorkItem 
| suspendWorkItem | ackSuspendWorkItem 
| launchCase | ackLaunchCase 
| updateWorkItemData | ackUpdateWorkItemData 
| forceCompleteWorkItem;

colset COREMSG = product CMD * PARAMS;

Fig. 6. A part of the colour sets

place, keeps the advice and the advised join point data. This token is consumed, in the second step, by the launchAdvice transition to launch the advice (fulfilling requirement R1.2). This transition is enabled when the data regarding the input parameters of the advice is retrieved, which is captured through a token with rspGetInputParams command in the ICore place. The third step is realized through the setAdviceState transition. This transition consumes the token in the joinPointID place. It also consumes a token from the ICore, which contains the ackLaunchCase command and the id of the launched advice. As a result, this transition produces a token in the ICore place with setAdviceInfo command.

The Pausing sub-net, depicted in Figure 8, is designed to address the pausing requirements. The net is enabled when the execution of a process in the WfMS reaches a Proceed placeholder (see Figure 3Ci). In this case, the WfMS raises a constraint, and the top-level module produces a token in the Proceed place, as described Section 4.1. The Pausing net should do three steps: i) pausing the advice (R2.1); ii) updating the advised join point data using the advice data; and iii) saving the relation between advice and Proceed placeholder, which is needed for resuming it later.

The first and third steps are started when the prepare transition produces tokens in the ICore place with a suspendWorkItem and saveJoinpointID commands correspondingly. However, the second step cannot be done until the advised join point id is known. Therefore, the prepare transition produces a token in the ICore place with getAdviceInfoByCase command to retrieve this in-
Fig. 7. CPN: Launching

formation. When this is done, the updateData transition consumes the token containing the result and starts the update by producing a token in the ICore place with updateWorkItemData command.

When all advices related to the advised join point are handled, and all data are synchronised, the service un-suspends the advised join point. Therefore, the setAdviceNum transition consumes all tokens which represent the advices of an advised join point and produces a token in the adviceNum place which contains the number of advices. The setAdviceInfo transition reduces the number of advices when the advice is suspended and the advised join point data is synchronised with the advice data. When the number of advices for an advised join point in the adviceNum place is zero, the proceedAdvisedJoinPoint transition gets enabled. This transition un-suspends the advised join point (R2.2). It also produces two tokens in the ICore place containing resumeFlag and getAdviceNumber commands. These tokens are used for resuming the advices.

The Core sub-net is responsible for sending messages to the WfMS as commands. The engine transition represents this communication (see Figure 9). It consumes tokens which represent messages to be sent to the WfMS. Consequently, it simulates the result of the interaction through tokens representing acknowledgements, which are produced in the ICore place. The rest of the net is designed to persist data among different steps of the weaving.

The ICore place in the Core module is both input and output place of the net. It supports the interaction between different modules. We describe this interaction as an example. As explained in Pausing module, the module should update the data of advised join point using advices’ data. Therefore, the updateData transition in the Pausing module produces a token in the ICore place, which
contains `updateWorkItemData` command (see bold arc from `updateData` transition to the `ICore` place in Figure 8). In fact, this is an external message that the `Pausing` module sends to the `Core` module. This token is consumed by the `engine` transition in the `Core` module. This transition simulates the WfMS. Thus, it produces a token containing the acknowledgement or the response of the command in the `ICore` module, if the command is not asynchronous. As a case in point, this transition produces a token with `ackUpdateWorkItemData` command in the `ICore` place as a result of consuming a token with `updateWorkItemData` command (see CMD colour set in Figure 6 for other commands). This token is used by the `Pausing` module, when it reduces the number of advices that are handled (see bold arc from the `ICore` place to the `setAdviceInfo` transition in Figure 8).

In addition to the engine related commands which are used to simulate the interaction of the service with the WfMS, there are some internal commands which are used for persisting data among different modules (see the non bold commands in the CMD colour set in Figure 6). We explain how they are utilized for internal communication through an example. As mentioned earlier, the `prepare` transition in the `Pausing` module produces a token in the `ICore` place, which contains `getAdviceInfoByCase` command (see Figure 8). In fact, this is an internal message that the `Pausing` module sends to the `Core` module. This token is consumed by the `getAdviceInfo` transition in the `Core` module. This transition consumes this token and a token from the `adviceInfo` place. As a result, it produces the same token in the `adviceInfo` place and the result of the internal command in the `ICore` place. The token which is produces in the `ICore` place contains the `rspGetAdviceInfoByCase` command and the information of the advice as parameters. This token is used by the `Pausing` module to pause the advice.

Due to space limitation, the remained nets are not described, but the same reasoning was followed when designing them.
Fig. 9. CPN: Core
5 Analysis

The functionality of a CPN Model can be investigated using simulations or state space analysis. There are two types of simulations, interactive and automatic. Interactive simulations facilitate investigation of the functionality of a system using a step by step analysis of the model. Automatic simulations are typically used to analyse the performance of the system [17] and are not relevant for our work. During the design, we used interactive simulation in order to continuously test the behaviour of the model and build it iteratively.

It should be noted that a simulation does not investigate all possible states of the system and hence cannot be used for verifying the behaviour of a model. All possible reachable states of a system are investigated through a state-space analysis [17], which is also the analysis technique we used for verifying the nets. We applied state-space analysis on a number of scenarios.

The scenarios were designed to capture all possible behaviour that can be encountered for the Aspect Service. We started with looking at the interplay of the number of aspects and advices in a model. Because the weaving is done using the main process and related advices, the number of advices influences the behaviour of a model, while the aspects alone do not influence it. As mentioned earlier, there are two types of advices, implicit or explicit, which need to be considered. Hence, four combinations of them can be derived: i) scenarios which contain implicit advices; ii) scenarios which contain implicit and explicit advices; iii) scenarios which contain explicit advices; and iv) scenarios in which the main process is not altered by any advices, i.e. either the advices are not considered since the pointcuts are not fulfilled, or there are not any advices defined.

To verify the behaviour of the model, the synchronisation of advices is essential. To study the synchronisation, at least two advices are needed. However, the process to synchronize advices to the main process is not changed by adding more advices to an advised joint point. Thus, two advices are both necessary and sufficient to verify the model.

As a result, we define the following four scenarios: Scenario A, containing two explicit advices; Scenario B, including one explicit and one implicit advices; Scenario C, containing two implicit advices; and Scenario D, describing the situation in which the main process is not altered by any advices, i.e. either the advices are not considered since the pointcuts are not fulfilled, or there are not any advices defined.

To verify the behaviour of the model, the synchronisation of advices is essential. To study the synchronisation, at least two advices are needed. However, the process to synchronize advices to the main process is not changed by adding more advices to an advised joint point. Thus, two advices are both necessary and sufficient to verify the model.

As a result, we define the following four scenarios: Scenario A, containing two explicit advices; Scenario B, including one explicit and one implicit advices; Scenario C, containing two implicit advices; and Scenario D, describing the situation in which the main process is not altered by any advices, i.e. either the advices are not considered since the pointcuts are not fulfilled, or there are not any advices defined.

To verify the behaviour of the model, the synchronisation of advices is essential. To study the synchronisation, at least two advices are needed. However, the process to synchronize advices to the main process is not changed by adding more advices to an advised joint point. Thus, two advices are both necessary and sufficient to verify the model.

As a result, we define the following four scenarios: Scenario A, containing two explicit advices; Scenario B, including one explicit and one implicit advices; Scenario C, containing two implicit advices; and Scenario D, describing the situation in which the main process is not altered by any advices, i.e. either the advices are not considered since the pointcuts are not fulfilled, or there are not any advices defined.

The Aspect Service interacts with a WfMS. This interaction is captured through sets of tokens representing the raising of constraints by the WfMS. The raising of the constraints follows a specific order (recall the four steps in Figure 3Ci). Therefore, we added a number of transitions and places in the model for each scenario to force the model to be executed according to the defined order. The results from the state space analyses for the four scenarios are shown in Figure 10.

As can be seen in the figure, the values for the Home Marking and the Dead Marking are the same for each scenario. This shows that the model is terminated in a specific state [16]. The state space reports show that there are not any live
transition instances and no infintive occurrence sequences. Hence, the model is sound.

Furthermore, there are some Dead Transition Instances for each scenario (not captured in the figure). Dead Transition Instances indicate parts of a model that are redundant and can be removed. However, the intersection of Dead Transition Instances is empty, i.e., $D_A \cap D_B \cap D_C \cap D_D = \emptyset$, where $D_X$ denotes the set of Dead Transition Instances of scenario X. This means that there are not any transitions in the model which are not used by any scenario. Therefore, the model does not have any redundancy.

Finally, a minor comment is that the number of nodes and arcs in the state space and the Strongly Connected Component (SCC) graphs are equal for each scenario. A Strongly Connected Component (SCC) indicates nodes which are mutually reachable (i.e., the nodes that can be reached from each other) and
reveals loops in a state space graph. The nodes which are not mutually reachable to other nodes (including themselves) are called trivial Strongly Connected Components (trivial SCCs). Since the number of nodes and arcs in the SCC graphs are equal to the number of nodes and arcs in the state space graph all nodes in the state space graph are trivial SCCs, for each scenario. Therefore, there is not any cycle in the CPN Model, which is according to the expectation since the weaving process do not contain any cycles.

Fig. 11. Part of the state space graph for Scenario A

We validated that the model fulfils the specified requirements through observations in the state space graphs. For instance, requirement R1.1 is realized because independently of execution path, the advised join point is always suspended. As depicted in Figure 11, one of the states 29, 30 and 32 (see the bold states in the figure) is reached during execution of the net. These states contain a token with a ackSuspendWorkItem command, which means that the advised join point is suspended. Similar observations were made for all the state space graphs of all four scenarios.

Finally, an aspect oriented business process model is sound if the main process model and related advice models are sound. The soundness is proved because the woven process is a composition of the main process and the models of the advices by using pairs of AND-splits and AND-joins (see Figure 3Cii) [4]. Therefore, the weaving process results always in a sound model.

6 Related Work

Aspect Orientation has long been an interesting subject in various Information Systems disciplines. A number of attempts have been made to adapt Aspect Orientation in different areas, among which are the Aspect Oriented Programming and the Aspect Oriented Business Process Management. The weaving is a key issue which is addressed when adapting Aspect Orientation for a particular
area. In this section we present some approaches on how the weaving has been addressed in the two areas listed above.

In the Aspect Oriented Programming, a formalization of the weaving is specified for AspectJ is specified in [10]. This work provides focuses on formalization to the static pointcuts. Moreover, the static weaving behaviour is investigated in [19]. This work facilitates the implementation of aspect orientation in Model Driven Engineering, as a case in point in code generators. It also investigates weaving multiple aspects in sequence diagrams. Furthermore some work has been done for adapting aspect orientation in Model Driven Development [18].

Despite many attempts which have been made for formalizing and modelling weaving, we could not find any related work in formalizing how the weaving should be performed in the BPM area. There is, indeed, an extension of Business Process Execution Language (BPEL) to support aspect orientation, which is called AO4BPEL [11]. However, this extension is language specific and does not show how the weaving should be performed at a general level. Furthermore, it does not specify the requirements which are necessary to implement weaving in WfMSs. Finally, it should be noted that BPEL does not support graphical representation of the business processes and BPEL does not support the involvement of human resources in running business processes, which are part of the Workflow Management Coalition Specification [25]. Thus, these limitations are applied to the AO4BPEL.

7 Conclusions and Future Work

In this paper, we presented how the weaving of aspects to business processes should be done. Our work provides a formalization of a service, called Aspect Service, which supports a WfMS to enact aspect oriented business process models. We based the formalization on Aspect Oriented Business Process Modeling Notation (AO4BPMN), which enables modeling processes and aspects separately. First, we presented the architecture of the Aspect Service. This architecture is founded based on the Worklet Service, which is designed to support flexibility in BPM area. Next, we defined the requirements based on which the service was designed using Coloured Petri Nets.

The nets were built iteratively using interactive simulations and verified using state space analysis. They were also validated by observing the behaviour of the model in state space graphs. Based on these analyses, we concluded that the solution is sound and fulfils the stated requirements.

Some direction for future work include: (i) an implementation of the service to support a WfMS; (ii) a comparison of Aspect Orientation in the programming and BPM areas. Such comparison would fortify Aspect Oriented BPM, as the Aspect Orientation is more mature in the programming area; (iii) applications of Aspect Oriented business process management in different areas, e.g., health care, bank and finance, to study benefits of the work in a real life setting; (iv) definition of a pointcut language which captures other business process perspectives such as the resource perspective; and (v) investigation on how the resource patterns [21], e.g., separation of duties and retain familiar, should be captured in orthogonal modularization.
Acknowledgement

We would like to thank Prof. Paul Johannesson for his valuable comments on earlier versions of the paper.

References