Extension of Petri Nets by Aspects to Apply the Model Driven Architecture Approach

E.E. Roubtsova [1]  
Open University of the Netherlands  
M. Aksit [2]  
University of Twente, Enschede, the Netherlands

Abstract
Within MDA models are usually created in the UML. However, one may prefer to use different notations such as Petri-nets, for example, for modelling concurrency and synchronization properties of systems. This paper claims that techniques that are adopted within the context of MDA can also be beneficial in modelling systems by using notations other than the UML. Petri-Nets are widely used for modelling of business and application logic of information systems with web services. For certain kinds of applications, therefore, Petri Nets can be more suitable for building Computation Independent, Platform Independent and Platform Specific Models (CIM, PIM and PSM). Unfortunately, the well-known problems with separation of concerns in Petri Nets and keeping track of changes may hinder achieving the aim of MDA: building reusable, portable and interoperable models. In this paper we define Aspect Petri Nets as a structure of several Petri Nets and quantification rules for weaving of those Petri Nets. Aspect Petri Nets are suitable for application of MDA; they support traceability of changes and reusability, portability and interoperability of models. We illustrate advantages of modelling in Aspect Petri Nets for MDA application and describe necessary tool support.

Keywords: Model Driven Architecture application, Petri Nets, Aspect-oriented system development, join point model, logic of weaving of aspects in Petri Nets, model transformations

1 Introduction
Model Driven Architecture (MDA) is attractive approach for system development. The main idea of MDA is to separate system specification from

[1] Email: Ella.Roubtsova@ou.nl
[2] Email: M.Aksit@ewi.utwente.nl

©2005 ABMB
Roubtsova, Aksit

the details of its implementation platform \[10\] and develop three groups of models: Computation Independent (CIM), Platform Independent (PIM) and Platform Specific Models (PSM) \[10\]. The transformation steps from CIM to PIM and from PIM to PSM should be specified as MDA mapping to guarantee traceability of design decisions and model reuse for other platforms and other systems \[6\].

Usually the models are created in the Unified Modelling Language (UML), the standard language of MDA \[9\]. However, in practice other graphical design notations are also widely used by different communities. The Model Driven Architecture approach can be equally applied when developing systems in notations different from the UML. One of these widely spread modelling techniques is the Petri Net technique. It exists since 1962 \[11\] and during last ten years this notation has extended its application domain. It is widely used for design of distributed systems, business application logic and information systems with web services \[14\]. An extended variant of Petri Nets namely Coloured Petri Nets \[7\], that allows specifying data, is suitable for modelling of CIM, PIM and PSM of different complexity. It possible to define the rules of transformation from CIM to PIM and from PIM to PSM as rules of Petri Net transformation.

Using Petri Nets for creating models for certain applications is attractive but there are shortcomings. Having several years of experience in business and industrial modelling using Coloured Petri Nets in master projects of our students \[3,5\] we identify the shortcomings as follows:

- Petri Nets have a problem with separation of concerns. Adding non-localisable concerns during model transformations from CIM to PIM and from PIM to PSM usually results in unreadable and spaghetti-like Petri Nets. This phenomenon is especially annoying for designers if the model does not fit within the screen of a computer.
- Petri Nets do not support keeping track of modifications. This shortcoming makes it difficult to reuse models, build them portable and interoperable, because models do not allow tracing design decisions without additional documentation.

The attempts to solve the problem with separation of concerns in Petri Nets by means of extensions by colours \[7\] and hierarchy do not give the expected result because colours and hierarchy hide concerns inside functions and hierarchical transitions correspondingly and make separation of concerns even more difficult.

It is well known that the object-oriented approach does not solve all the problems with separation of concerns in Petri Nets \[12\].

Having experienced problems with Petri Net modelling in practice, we set for ourselves the task to improve readability, traceability and reusability of Petri Nets in order to make them suitable for application of the MDA approach. To separate concerns in Petri Nets, we define Aspect Petri Nets
as a structure of several Petri Nets and introduce a join point model in Petri Nets. On the basis of the join point model we propose mechanisms for static weaving Petri Nets together. To avoid one dimension of complexity when introducing new ideas we restrict ourselves by classical Petri Nets, a subset of Coloured Petri Nets.

The paper is organized as follows. Section 2 shows problems with separation of concerns in Petri Nets. In Section 3 we define Aspect Petri Nets. In Section 4 we propose a join point model and a language for weaving Petri Nets together. Section 5 illustrates the usage of the language by examples of weaving expressions and Petri Nets constructed in correspondence with those expressions. Section 6 concludes the paper by indicating advantages of Aspect Petri Nets for MDA application. Future work is also discussed in this Section.

2 Problems with separation of concerns in Petri Nets

In this section we give simplified examples of a computation independent model, a platform independent model and a platform specific model represented in classical Petri Nets and show the shortcomings of Petri Nets for MDA application.

CIM. Let us consider an Internet shop where a client is able to look at offers, make his choice and purchase the chosen goods. A CIM of the Internet shop shown in Figure 1 represents the system from the client point of view. A token in the place client represents a client. One token in the place instance restricts the number of clients by one. The actions that a client can fulfill using the shop are the following:

- a client can look for a specific good (transition look);
- he/she can make a choice of a good (transition choose);
- then he/she can repeat searching and choosing (transition more)
- or pay for goods (transition pay) and leave the internet-shop.

![Fig. 1. A CIM of an Internet shop](image)

PIM. For the sake of simplicity our platform independent model contains only two computation dependent concerns: logging all user initiated events and cancelling work by a user.
The logging concern is non-localisable. Every transition initiated by a user produces an output place log: log1, log2, log3, log4 (Figure 2).

The concern of cancelling presents an opportunity to cancel the process of shopping at any intermediate point and return the system into its initial state. The intermediate points are modelled by the places choice and chosen. The initial state of the system is presented in the Petri Net Service by one token depicted by a black bullet in place client and one token in place instance: (client = 1, instance = 1).

PSM. A platform specific model should include architectural elements. Now assume that we would like to use the CORBA platform that performs dynamic service selection and invocation [1] Our Internet shop application requests: a search service before transition look and a payment service before transition pay.

Transition request1(request2) models a request of a service and produces a service into its output place service1 (service2). If a service has not been found then the control is taken by transition deny1 (deny2). If a service has been found then transitions permit1 (permit2) takes control.

Analysing even our simplified example of PSM (Figure 2) we can see that modelling several concerns worsens readability of the model. If we model all the concerns taken into account in industrial applications, then we shall lose the readability of our model completely. The track of design decisions in the PSM is lost, i.e. design decisions cannot be recognized without additional description. It is difficult to trace the design decisions back and reuse intermediate models for another platforms or applications.

To solve the problem of losing readability and traceability of design decisions when modelling in Petri Nets we propose a new notation named Aspect Petri Nets. This notation uses Petri Nets to model aspects separately and a logical language to specify rules of aspect weaving. A concern is modelled by weaving several aspects according to the specified weaving rules. The specification of weaving rules allows backtracking design decisions. Weaving rules of
a concern are used to visualise the concern model in form of the corresponding Petri Net. Weaving rules allow constructing simulation models.

3 Aspect Petri Nets

3.1 General Principles of Aspect-Oriented Approach to Software Development

The aspect-oriented approach has been successfully applied for creating aspect-oriented programming languages [2]. The aspect-oriented approach to software development defines an aspect as a unit designed to implement a concern that cannot be localized [2]. To build an aspect-oriented approach based on a chosen notation one should find a join point model and an aspect quantification mechanism for this notation. A join point model defines the elements in a model where aspects can be attached. The join point model depends on the form of event and state presentation in the chosen notation. The join point model restricts the allowed quantification or weaving expressions, i.e. the types of predicates that a designer can use to attach aspects to each other. A concern is represented by an aspect together with its weaving expression.

One of the main principle of the aspect-oriented approach is the principle of obliviousness of concern specification, which means that the concern on which we quantify should not know about other concerns and the mechanisms used for their quantification. Obliviousness allows designers to produce independent specifications of aspects.

3.2 An Aspect Petri Net

Let us define an Aspect Petri Net follow the general principles of the aspect-oriented approach to software development.

A classical Petri Net

is a tuple \( N = (P, T, F, M^0) \), where \( P \) is a finite set of places, \( T \) is a finite set of transitions, \( F \subseteq (P \times T) \cap (T \times P) \) is the set of arcs, called a flow relation, \( M^0 \) is the initial marking \( P \rightarrow \{0, 1, 2, 3, ... \} \) [8].

We define an Aspect Petri Net

as a triple \( A = (SN_1, SN_2, D(SN_1, SN_2)) \) where

- \( SN_1 \) is a set of Petri nets to which we join an aspect;
- \( SN_2 \) is a set of Petri nets specifying an aspect;
- \( D(SN_1, SN_2) \rightarrow \{true, false\} \) is a logical expression, named a designator or a weaving expression [2], that describes where the nets of set \( SN_2 \) can be invoked and how to join them to the nets of set \( SN_1 \).
A classical Petri Net $N$ is an Aspect Petri Net $A = (\emptyset, \{N\}, true)$ initialized only ones without any rules, such that $D(\emptyset, N) = true$.

An Aspect Petri Net such that sets $SN_1 = \{N_1\}$ and $SN_2 = \{N_2\}$ are singletons can be represented by an abstract UML class diagram shown in Figure 3: \texttt{PetriNet} and \texttt{Designator}. Stereotype \texttt{PetriNet} is used to represent a Petri Nets $N$ modelling an aspect. All places and transitions of a Petri Net are the attributes of an instance of this stereotype. Stereotype \texttt{Designator} is a classifier for the association type representing weaving of aspects. The association is unidirectional to guarantee obliviousness of concern specification. The operations of stereotype \texttt{Designator} shown in Figure 3 are defined in Section 4 on the basis of the join point model.

To let our readers evaluate the advantages of modelling in Aspect Petri Nets, we show in Figure 4 the PSM of our Internet shop modelled as an Aspect Petri Net. The aspects are represented as simple Petri Nets and design decisions are kept as designators also named weaving expressions. The PSM
is easy to read and understand. The track of design decisions is kept by the
designators. For the simulation purpose the weaving expressions are applied
as constructive commands to build simulation models. In Section 5 we present
the weaving expressions and their constructive semantics for our case study.
But first, let us define a language for specification of weaving expressions.

4 A Language for Static Weaving of Classical Petri Nets

There are two types of designation points in classical Petri Nets: places and
transitions. The sets of places and transitions of weaved Petri Nets form the
join point model for static weaving of classical Petri Nets.
Let name specifications for $< \text{netName} >$, $< \text{placeName} >$, $< \text{transitionName} >$
and $< \text{setName} >$ be given. For all definitions of next subsections we use the
following conventional names:
$< N >, < N_1 >, < N_2 > :: = < \text{netName} >$
$< e >, < e_1 >, < e_2 > :: = < p > \mid < t >$
$< p >, < p_1 >, < p_2 >, < p_2^a >, < p_2^b > :: = < \text{placeName} >$
$< t >, < t_1 >, < t_2 > :: = < \text{transitionName} >$

A pointcut designator for an Aspect Petri Nets must at least provide the
following basic operations, which are explained in the following subsections:

- Join (Disjoin) to Place and Join (Disjoin) to Transition operations which
  are used to attach (detach) Aspect Petri Nets to (from) each other;
- Insert (Remove) to Place and Insert (Remove) to Transition operations,
  which cut an Aspect Petri Nets and extend (reduce) this net.

4.1 Invocation a Petri Net

**Definition 4.1** $\text{invoke}( < N_1 > . < e >, < N_2 > )$. Let Petri Nets $N_1$ and
$N_2$ and a name of a designation point $N_1.e$ of net $N_1$ be given. Operation
$\text{invoke}(N_1.e, N_2)$ creates a copy $e.N_2$ of net $N_2$ and returns value $true$. The
names of all places and transitions of $e.N_2$ are extended by prefix $e$.

4.2 Join Operations

Informally a join operation merges two elements of different nets together,
so that the resultant element gets the union of the input arcs and the union
of output arcs of both initial elements. The merged elements should be of
the same type: a place is merged with a place, a transition is merged with a
transition.

**Definition 4.2** $\text{joinToPlace}( < N_1 > . < p_1 >, < N_2 > . < p_2 > )$. Let two
Petri nets $N_1 = (P_1, T_1, F_1, M_1^0)$ and $N_2 = (P_2, T_2, F_2, M_2^0)$ (Figure 5) be given
where
- $p_1 \in P_1, p_2 \in P_2$;
• $p_1$ is the set of input transitions of place $N_1.p_1$.
(We follow the traditional notation in Petri Nets [8])
• $p_1 \bullet$ is the set of output transitions of place $N_1.p_1$.
• $p_2$ is the set of input transitions of place $N_2.p_2$.
• $p_2 \bullet$ is the set of output transitions of place $N_2.p_2$.
• $(\{(t, p_1)| t \in p_1\} \cup \{(p_1, t)| t \in p_1 \bullet\}) \in F_1$.
• $(\{(t, p_2)| t \in p_2 \\cup \{(p_2, t)| t \in p_2 \bullet\}) \in F_2$.

![Fig. 5. Operations joinToPlace() and joinToTransition()](image)

Operation $\text{joinToPlace}(N_1.p_1, N_2.p_2)$ creates net $N_3$ and returns value true.
The net $N_3 = (P_3, T_3, F_3, M_3^0)$ (Figure 5) is the following:

• $P_3 = P_1 \setminus \{p_1\} \cup P_2 \setminus \{p_2\} \cup p_1.p_2$,
  where $p_1.p_2$ is a new place with name $p_1.p_2$.
• $N_3.p_1.p_2 = \bullet N_1.p_1 \cup \bullet N_2.p_2$ is the set of input transitions of $N_3.p_1.p_2$.
  Each transition $t \in N_2.p_2$ is renamed in $N_3$ to $p_1.t$.
• $N_3.p_1.p_2 \bullet = N_1.p_1 \cup N_2.p_2$ is the set of output transitions of $N_3.p_1.p_2$.
  Each transition $t \in N_2.p_2 \bullet$ is renamed in $N_3$ to $p_1.t$.
• $T_3 = T_1 \setminus (\bullet p_1 \cup p_1 \bullet) \cup T_2 \setminus (\bullet p_2 \cup p_2 \bullet) \cup (\bullet p_1.p_2 \cup p_1.p_2 \bullet)$.
• $F_3 = F_1 \setminus (\{(t, p_1)| t \in p_1\} \cup \{(p_1, t)| t \in p_1 \bullet\}) \cup F_2 \setminus (\{(t, p_2)| t \in p_2 \\cup \{(p_2, t)| t \in p_2 \bullet\}) \cup \{(t, p_1.p_2)| t \in p_1.p_1 \cup \{(p_1, p_2, t)| t \in p_1.p_2 \bullet\}.
• $M_3^0$: The markings of all places of $P_1$ and $P_2$ are the same. The marking of place $p_1.p_2$ is the same as the marking of place $p_1$. 

8
Definition 4.3 \textit{joinToTransition}(< N_1 > . < t_1 >, < N_2 > . < t_2 >).

Let two Petri nets \(N_1 = (P_1, T_1, F_1, M_1^0)\) and \(N_2 = (P_2, T_2, F_2, M_2^0)\) be given, such that

- \(t_1 \in T_1; t_2 \in T_2;\)
- \(\bullet t_1\) is the set of names of input places of transition \(N_1.t_1\).
- \(t_1\bullet\) is the set of names of output places of transition \(N_1.t_1\).
- \(\bullet t_2\) is the set of names of input places of \(N_2.t_2\).
- \(t_2\bullet\) is the set of names of output places of \(N_2.t_2\).

- \(\{(p, t_1) | p \in \bullet t_1 \} \cup \{(t_1, p) | p \in t_1\bullet\}\) is a flow relation of net \(N_1\).
- \(\{(p, t_2) | p \in \bullet t_2 \} \cup \{(t_2, p) | p \in t_2\bullet\}\) is a flow relation of net \(N_2\).

Operation \textit{joinToTransition}(N_1.t_1, N_2.t_2) creates net \(N_3 = (P_3, T_3, F_3)\) and returns value \textit{true}. The net \(N_3\) is created as follows:

- \(P_3 = P_1 \cup P_2;\)
- \(T_3 = T_1 \setminus \{t_1\} \cup T_2 \setminus \{t_2\} \cup \{t1.t2\},\) where \(t_1.t_2\) is a transition named \(t_1.t_2\).
- \(\bullet t_1.t_2 = \bullet N_1.t_1 \cup \bullet N_2.t_2\) is the set of input places of \(N_3.t_1.t_2\);
  Each place \(p \in \bullet N_2.t_2\) is renamed in \(N_3\) to \(t_1.p\).
- \(t_1.t_2\bullet = N_1.t_1 \cup N_2.t_2,\) is the set of output places of \(N_3.t_1.t_2\);
  Each place \(p \in N_2.t_2\) is renamed in \(N_3\) to \(t_1.p\).
- \(F_3 = F_1 \setminus (\{(p, t_1) | p \in \bullet t_1 \} \cup \{(t_1, p) | p \in t_1\bullet\} \cup \{(p, t_2) | p \in \bullet t_2 \} \cup \{(t_2, p) | p \in t_2\bullet\})\) \cup \(\bullet P_1.t_2 \cup t_1.t_2\bullet\).
- \(M_3^0 = M_1^0 \cup M_2^0\).

The definitions of the reverse operations \textit{disjoinFromPlace}(N_3,p_1.p_2, N_2.p_2) and \textit{disjoinFromTransition}(N_3.t_1.t_2, N_2.t_2) are straightforward and illustrated by the same Figure 5.

4.3 Insert Operations

Insert operations cut the initial net.

Definition 4.4 \textit{insertToPlace}(< N_1 > . < p_1 >, < N_2 > . < p_2 >, < N_2 > . < p_2 >)

Operation \textit{insertToPlace}(N_1.p_1, N_2.p_2, N_2.p_2) creates net \(N_3\) and returns value \textit{true}. Net \(N_3 = (P_3, T_3, F_3, M_3^0)\) is the following:

- \(P_3 = P_1 \setminus \{p_1\} \cup P_2 \setminus \{p_2, p_2^b\} \cup \{p_1.p_2, p_1.p_2^b\};\)
- \(T_3 = T_1 \cup T_2;\)
- \(F_3 = F_1 \setminus (\bullet N_1.p_1 \cup N_1.p_1\bullet) \cup F_2 \cup \{(t, p_2^b) | t \in \bullet N_1.p_1 \} \cup \{(p_2^b, t) | t \in N_1.p_1\bullet\},\)
- \(M_3^0\): Places from sets \(P_1\) and \(P_2\) keep their markings.
  Place \(p_1.p_2\) has the marking of place \(p_1\), place \(p_1.p_2^b\) has the empty marking.

Operation \textit{deleteFromPlace}(N_3,p_1*, N_2.p_2, N_2.p_2) is the reverse operation to the operation \textit{insertToPlace}(N_1.p_1, N_2.p_2, N_2.p_2^b).
The principle of symmetry leads us to a definition of operation \( \text{insertToTransition}(N_1.t_1, N_2) \). This operation is used in hierarchical Petri Nets \([7]\). The correspondence between all input places of \( N_1.t \) and input places of \( N_2 \) and the output places of \( N_1.t \) and output places of \( N_2 \) should be specified for this operation. Net \( N_2 \) becomes hidden inside transition \( t \). The operation could be used for weaving of a specific aspect, inputs and outputs of which match to inputs and outputs of several transitions.

### 4.4 Predicates for the language of aspect weaving

The boolean expressions corresponding to operations defined above are the predicates of the language of aspect weaving:

\[
< \text{Invoke}(< N_1 > . < e_1 > . < N_2 >) >::= \\
\text{invoke}(< N_1 > . < p_1 > . < N_2 >) | \text{invoke}(< N_1 > . < t_1 >, < N_2 >); \\
< \text{Expr}(< N_1 > . < e_1 >, < N_2 >, < e_2 >) >::= \text{true} | \text{false} | \\
\text{joinToPlace}(< N_1 > . < p_1 >, < N_2 > . < p_2 >) | \\
\text{joinToTransition}(< N_1 > . < t_1 >, < N_2 > . < t_2 >) | \\
\text{insertToPlace}(< N_1 > . < p_1 >, < N_2 > . < p_a >, < N_2 > . < p_b >) | \\
< \text{Expr}(< N_1 > . < e_1 >, < N_2 >, < e_2 >) \land \\
< \text{Expr}(< N_1 > . < e_1 >, < N_2 >, < e_2 >) | \\
< \text{Expr}(< N_1 > . < e_1 >, < N_2 >, < e_2 >) \lor \\
< \text{Expr}(< N_1 > . < e_1 >, < N_2 >, < e_2 >).
\]

### 4.5 Weaving expressions of the language for weaving aspects

A weaving expression of the language for aspect weaving is a quantifier over elements of a given net. It has a boolean value. A weaving expression is constructive in the sense that it presents an algorithm of weaving aspects modelled by Petri Nets.

A weaving expression can be specified as follows:
The quantifier $\langle WE \rangle$ means "for all elements $e_1$ of net $N_1$ such that the boolean expression $B(N_1, e_1)$ is true net $N_2$ representing an aspect is invoked and joined (inserted) to element $N_1.e_1$ such that $Expr$ is true".

Some useful quantifies are composed from the operations defined above, for example, $\text{insertAfterTransition}()$ and $\text{insertBeforeTransition}()$.

**Definition 4.5** Operation $\text{insertAfterTransition}()$

$\text{insertAfterTransition}(N_1.t_1, N_2.t_2, N_2.p^a, N_2.p^b) ::= \forall p: p \in N_1.t_1 \bullet \left[ \text{invoke}(N_1.p, N_2) \land \text{insertToPlace}(N_1.p, N_2.p^a, N_2.p^b) \right].$

**Definition 4.6** Operation $\text{insertBeforeTransition}()$

$\text{insertBeforeTransition}(N_1.t_1, N_2.t_2, N_2.p^a, N_2.p^b) ::= \forall p: p \in N_1.t_1 \left[ \text{invoke}(N_1.t_1, N_2) \land \text{insertToPlace}(N_1.p, N_2.p^a, N_2.p^b) \land \text{joinToTransition}(N_1.t_1, N_2.t_2) \right].$

The general expression for a designator when quantifying on a set of nets is the following:

$\langle D \rangle ::= \forall N_1 \bullet \left[ \langle B(N_1) \rangle \land \langle WE \rangle \right];$

$\langle B(N_1) \rangle ::= \langle N_1 \rangle = \langle \text{netName} \rangle \lor \langle N_1 \rangle \in \langle \text{setName} \rangle;$

"for all nets such that predicate $B(N_1) = true$: $WE = true$".

## 5 Transformation of Models in Aspect Petri Nets

In this section we illustrate transformation from CIM to PIM and from PIM to PSM on the example of our Internet shop described in Section 2.

### 5.1 Logging concern

The logging concern is modelled by Petri Net $\text{Logging}$ (Figure 7) that consists of only one transition $write$ and one place $log$ with an arc from $write$ to $log$. Transition $write$ models writing to a log-file. Place $log$ collects the results of logging.

To weave the logging concern and the CIM of the Internet shop shown in Figure 1 we use the following weaving expression:

$D(\text{Service, Logging}) : \forall \text{Service}.t : [ \text{invoke}(\text{Service}.t, \text{Logging}) \land \text{joinToTransition}(\text{Service}.t, \text{Logging}.write) ];$

The expression in our language is constructive, i.e. it describes an algorithm of weaving:
Let nets \textit{Service} and \textit{Logging} are given. For all transitions \( t \) of net \textit{Service} repeat:

1. Make a copy \( t.\text{Logging} \) of net \textit{Logging} and extend names of all its elements by prefix \( t \).
2. Join transition \( t \) to transition \( t.\text{write} \) of net \( t.\text{Logging} \);

Figure 7 shows the result of weaving the logging aspect with net \textit{Service} according to the defined designator. Each firing of transition \textit{Service}.\( t \) produces a log-record, modelled by the corresponding place \( t.\text{log} \).

![Diagram of Petri Nets Service and Logging](image)

Fig. 7. Concern of logging of events.

### 5.2 Cancelling concern

The cancelling aspect is modelled by Petri Net \textit{Cancel} shown in Figure 8. This net has only one transition \textit{Cancel}. This transition has one input place named \textit{input} and two output places \textit{initial} and \textit{capacity}. The weaving of Petri Net \textit{Cancel} with Petri Net \textit{Service} is specified by the designator:

\[
D(\text{Service}, \text{Cancel}) : \\
\forall \text{Service}.p : (\text{Service}.p = \{\text{choice}, \text{chosen}\}) \\
[ \text{invoke}(\text{Service}.p, \text{Cancel}) : \text{joinToPlace}(\text{Service}.p, \text{Cancel}.\text{input}); \\
\text{joinToPlace}(\text{Service}.\text{client}, \text{Cancel}.\text{initial}) \land \\
\text{joinToPlace}(\text{Service}.\text{instance}, \text{Cancel}.\text{capacity})];
\]

The weaving procedure defined by this expression is the following:

For places \( p = \text{choice} \) and \( p = \text{chosen} \) repeat:

1. Make a copy \( p.\text{Cancel} \) of net \textit{Cancel} and extend names of all its elements by prefix \( p \).
2. Join place \( p \) to place \( p.\text{input} \) of net \( p.\text{Cancel} \); place \textit{client} to place \( p.\text{initial} \); place \textit{instance} to place \( p.\text{capacity} \).

Figure 8 represents the result of weaving according such a designator. Each place of the former net \textit{Service} gets an alternative to return the net into its initial marking. Nets \textit{Cancel} and \textit{Service} are traceable in the result of weaving.
The result of weaving Petri Nets Service and Cancel

Petri net Cancel

Fig. 8. Concern of cancelling.

5.3 PSM in Aspect Petri Nets

Fig. 9. Service Request.

The PSM contains the service request concern. A service is requested for two transitions: look and pay. An aspect of request is modelled by net Request shown in Figure 9. The model of the aspect is reusable. The weaving expression for this aspect uses the operation insertBeforeTransition():

\[ D(\text{Service}, \text{Request}) \]
∀Service.p : (Service.p ∈ •Service.t₁ ∧ Service.t₁ = {look, pay})
    \[ \text{invoke}(Service.p, Request) \land \\
    \text{insertBeforeTransition}(Service.t₁, Request.permit, Request.in, \\
    Request.service) \].

The weaving procedure defined by the weaving expression is the following:

For transitions $t=\text{look}$ and $t=\text{pay}$ repeat:

1. Make a copy $t.Request$ of the aspect net $Request$;
2. For all input places $p \in •t$ of transition $t$:
   2.1. split place $p$ in two places: $p^a$, $p^b$, such that input arcs belong to place $p^a$ and output arcs belong to place $p^b$;
   2.2. join place $p^a$ to place $t.Request.in$, join $p^b$ to $t.Request.service$;
3. Join transition $t$ to transition $t.Request.permit$;

The result of weaving is shown in Figure 9. This model can be used for simulation of the concern $Request$.

6 Conclusion and Future Work

The MDA approach to system modelling provides obvious advantages for designers and companies: it allows building reusable models and portable applications. MDA can be successfully applied to different design notations used in specific fields, however the mechanisms of model transformations in those notations should guarantee separation of concerns and traceability of design decisions. The novel contribution of this paper is the extension of Petri Nets by aspects and weaving mechanisms for aspects. Our new notation named Aspect Petri Nets is suitable for using in the MDA context due to at least three reasons:

- Firstly, bringing the advantages of aspect-orientation and MDA to Petri Nets provides new perspectives to the applications where Petri nets are widely used.
- Secondly, in certain cases, transformations from one model to another can be adequately represented as aspect weaving operations. This was illustrated by the example in adding computation dependent and CORBA specific features in the model.
- Thirdly, as illustrated in the example, the design decisions represented by weaving expressions make models more traceable, retrievable and reusable.

The proposed mechanisms and their advantages, therefore complement the other known benefits provided by the MDA approach, and make MDA even more attractive while using Petri Net like formalisms.

We are implementing tool support for MDA approach to modelling in Aspect Petri Nets. To support designers in writing weaving expressions, an expression builder is designed to provide the lists of nets and operations and to make syntax checks using the grammar of the weaving language defined.
in this paper. We are implementing a module for specification and verification of weaving correctness and a module for constructing simulation models using weaving rules. Investigation of join point model and dynamic weaving mechanisms for Coloured Petri Nets is considered as future work.

References


