Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 189 (2007) 69–85
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Safe Adaptation of Component Coordination

Christophe Sibertin-Blanc[1]

Institut de Recherche en Informatique de Toulouse, UMR CNRS 5505
Université Toulouse 1, 1 Place Anatole France, F-31042 Toulouse Cedex

Philippe Mauran, Gérard Padiou[2]

Institut de Recherche en Informatique de Toulouse, UMR CNRS 5505
ENSEEIHT, 2 rue Camichel, BP 7122, F-31071 Toulouse cedex 7

Abstract

In the domain of software engineering, the use of software components is now a well established approach. However, it raises problems about the dynamic adaptation of these components to particular users demands. Indeed, these components have been developed with the intent to have a wide range of use, and so they implement functionalities which perhaps do not match precisely enough the demands of specific users.
Therefore, we address the adaptation of the coordination between components by means of so-called Moderators. A Moderator is itself a coordination component managing interactions that are described and formalized using Petri nets. More precisely, we study the dynamic adaptation of the coordination rules by means of specific transformations of the Petri nets used to describe a Moderator.
Safety properties must be enforced to maintain a consistent cooperation among participants with respect to the requested evolutions of the coordination rules. In particular, an adaptation of the Moderator can be considered safe if it cannot be detected by the participants. We present a computable criterion which enables to check such a satefy property automatically.
We illustrate our approach in the context of a computer aided learning system, by adapting the coordination rules for controlling accesses to documents during an examination.

Keywords: coordination, adaptation, software components, Petri nets, finite automata, safety criterion.

1 Introduction

Software reuse is an old and essential concern in the field of software engineering. In this search, the notions of interface and modularity have progressively emerged, leading to the current notion of software component [9,11]. Composing, assembling components is the core of software architectures. Component models are based on two key features:

[1] Email:sibertin@univ-tlse1.fr

[2] Email:mauran@enseeiht.fr, padiou@enseeiht.fr

doi:10.1016/j.entcs.2007.05.048

- the specification of composition mechanisms or connectors [16,1]: in this case, behavioral aspects play an important role, in addition to structural aspects [8,20].
- the ability to reuse, and thus the necessity to adapt existing components [5,12].

Inasmuch as a component is to be widely reused, the designer of this component cannot consider in advance every possible context of use for this component. The underlying idea to the component approach is to provide the means to adapt, and customize a "standard" code, in order to bridge the gap between the "intended" use of a component (at the time of its design), and its effective use.

Such a gap appears when a component is to be combined with other components, in other words when a component is to interact with other components. Thus, this mismatch can be reduced by altering either the component itself, or the interaction, the protocol which defines the composition of the component with its current environment. The bulk of the works in the field of Software Adaptation [6] focuses on the structural aspects of composition, and use components' interfaces as a basis for defining and implementing adaptations. The interaction patterns considered in this approach are elementary ones, namely (synchronous) procedure calls, and (asynchronous) events publishing/subscribing. However, several works do focus on adapting interactions rather than components, and thus explore behavioral aspects, and the support to making up, and adapting more complex interactions.

This non invasive approach appears to be fitted for handling components as black boxes. In this setting, interactions and their adaptation can either be defined by characterizing the (possible) participating components, as in [2,18] or by focusing more specifically on the interaction protocol itself, as in [6,19], and this paper. Moreover, the support to implementing adaptations is developed along two main lines:

- either generate (semi-)automatically
  - · adaptors from the specification of the participants' behaviors [20,6,19], or
  - · the participants' behavior specification, from the interaction specification [18];
- or provide the programmer with tools that support the checking of the correctness of adaptations. This approach is illustrated in the domain of the choreography of Web Services in [2].

Our approach comes within the latter line, and focuses on the adaptation of interactions: we consider the possibility of allowing adaptations of the coordination specified in the general framework proposed in [10]. In this framework, coordination is achieved by a specific component, called a Moderator, that controls the sequence and conditions, i.e. the choreography [4], in which interactions are performed. The specification of the coordination is considered from the designer's perspective: the possible uses of a component are characterized by roles, whose compliance with the component's semantics is checked a priori.

The main contribution of this paper is to define a computable safety criterion, which allows to determine whether an adaptation is correct, without having to intervene into the semantics either of the protocol or of the participants. This appears to be interesting in the context of open systems, where programs are built by assembling preexisting components. This assembly is made possible by using and maintaining invariant relationships between the assembled components. The notion of interface thus allows to abstract and encapsulate the implementation of components. The adaptation of coordination (and our approach in particular) extends this process by aiming at enabling to alter the actual behavior of a set of coordinated components, while keeping invariant the contract (the specification of the coordination) that binds these components altogether. More precisely, a participant may ask the Moderator of a conversation to depart from the (preset) behavioral rules of the protocol, in order to adapt to a specific runtime context. To be safe, such an adaptation should guarantee each participant taking part to the conversation that the goal of the conversation can be reached.

In the following sections, we first describe the functionalities and the properties of coordination components. Then, we define adaptations, the safety property related to these adaptations and how it can be proved. Lastly, we illustrate this adaptation approach through an e-learning case study.

2 Coordination Protocols

Protocols are intended to ensure coordination between entities of a system. A protocol is defined as a set of rules, and it is instantiated as processes, that we call *conversations*, in the course of which entities follow these rules to coordinate their respective behaviors. These rules determine which participants may take part in a conversation, and how each one can or must contribute to its good processing. In other words, a conversation is a process which proceeds according to a protocol defined by the following items [10]:

- *Data* that need to be processed in the course of a conversation,
- The *initial state* of a conversation, i.e. the conditions that must be satisfied so that a conversation can start,
- The *final state* that characterizes the completion of a conversation,
- The *roles* that participants can hold in a conversation, i.e. the specific contributions to the achievement of the protocol,
- *Casting constraints* on the attribution of roles that determine the conditions to satisfy so that a participant may take a certain role in a given conversation,
- The *types of interventions* that participants can carry out to take part in a conversation, to make it progress,
- The *behavioral constraints* that determine the control structure of the conversations, i.e. in which cases a participant playing a certain role can carry out a given intervention, as well as the effect of this intervention.

The next section, and the case study (Section 4) provide two examples of coordination protocols.

2.1 Moderators as coordination components

The main benefit of coordination by protocols is to ensure the efficiency and the predictability of interactions among participants with regard to the achievement of some task. However, whatever the rules of a protocol, each participant must be sure that they will be respected by others participants; otherwise, there is no guarantee that the common objectives of the protocol will be reached. This can be achieved by dissociating on the one hand the interventions in a conversation that are performed by the participants, and on the other hand checking whether these interventions obey to the protocol rules, which is performed by a specific component called the *Moderator* of the conversation [10]. The Moderator is in charge of enforcing the protocol rules by making the participants in the physical impossibility to contravene the rules of the protocol [7]. More precisely, a Moderator:

- checks at its creation whether the protocol's initial state is valid;
- records the required data into variables or into a database;
- upon a request from a participant to play some role in the conversation, returns a positive (or negative) answer according to the fulfilment of the casting constraints;
- ensures that the course of the conversation fulfils the protocol's behavioral constraints. To this end, the Moderator centralizes all the communications between the participants in the conversation: any intervention of a participant is directed to the Moderator and, if it is coherent with the current state of the conversation and the rules of the protocol, the Moderator accounts for and processes it by sending appropriate messages to other participants;
- decides the end of the conversation, after having detected either that the final state is reached, or that the conversation is blocked for some reason, e.g. the leaving of a participant whose contribution is essential to ending the conversation.

To implement the behavioral constraints of a protocol it is proposed in [10] to model the control structure of Moderators by means of the Petri net formalism [13]. The Petri net (in short PN) of a Moderator includes *communication places*: input places to receive the messages sent to the Moderator by the participants, together with output places for the messages from the Moderator to the participants. The initial marking of this net corresponds to the initial state of any conversation of the protocol and its terminal (deadlocking) marking corresponds to the final state.

Figure 1 presents a simple online auction web site. The Moderator, which is the shaded (central) part of the figure, mediates and controls the interactions between two kind of participants: customers (which are represented on the right-hand part of the PN), and vendors (left-hand part of the PN).

The basic scenario is the following: customers order goods from the website (which is implemented by the Moderator), and then send their payment to the website. Lastly, they acknowledge the receipt of their goods to the website. The Moderator forwards the orders and the payments that it receives to the vendors, and records the acknowledgments. Vendors accept orders (received from the

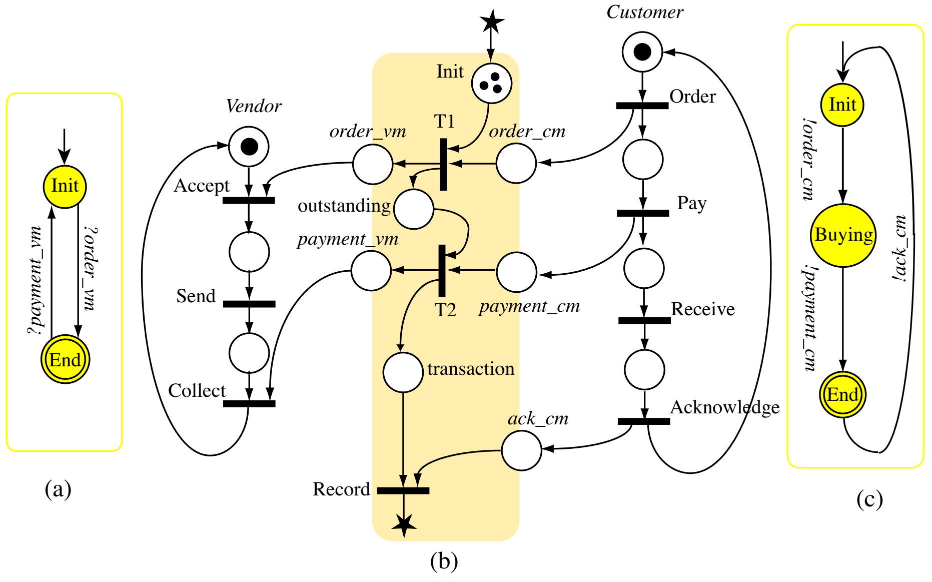


Fig. 1. Online auction website

Moderator), send goods, and then collect payments[3]. The behavior of vendors and customers is described by finite automata: a Vendor only receives messages and a Customer only sends messages. This PN includes 5 communication places ($order\_vm, payment\_vm, order\_cm, payment\_cm$ and ack_cm) that correspond to the types of messages exchanged between the Moderator and the participants.

A High Level Petri net formalism is used, in order to process data carried by tokens [17]. Notably, each message-token in a communication place includes the identity of the participant that has sent or shall receive this token. In this way, the Moderator is able to manage a dedicated thread of control for each participant and to keep track of its state with regard to the conversation.

The PN of the Moderator of a protocol is a formal definition of the protocol's behavioral constraints, since it determines, for any participant in the conversation, in which cases the Moderator can process a message received from this participant or send it a message.

Using the PN formalism to define the behavioral rules of a protocol allows to use techniques and tools of the PN theory to analyze behavioral properties. We do not elaborate on other aspects of Moderators (e.g. the management of data, of sessions, of threads of control, or granting roles to participants) since we are only concerned here by their behaviors.

2.2 The behavior of a role

The *Internal Petri Net* (IPN) of a Moderator is obtained from its PN by removing all the communication places. The sequences of transitions which are enabled in the

[3] We do not represent an interaction which is out of the Moderator's scope, namely the (physical) transmission of goods: a complete system should have a place linking the Vendor's Send and the Customer's Receive transitions, but this is irrelevant for our purposes, as this is not managed by the Moderator.

Moderator's IPN at the initial state are exactly the sequences of transitions that may occur during a conversation in the course of which each participant behaves according to its role. In particular, this net assumes that every message sent is used, and that any expected message will finally be emitted.

We assume that the IPN of a Moderator features the proper termination property [13]: the final state is the only deadlock state, and it can always be reached from any state that is reachable from the initial state. This property states that, in any case, the protocol's behavioral rules effectively enable the coordinated components to reach the goal of the protocol. This property is decidable, as it can be expressed as the liveness of a final transition.

We also assume that the IPN of a protocol is a bounded Petri net, that is there is an upper bound to the number of tokens in any marking that is reachable from the initial marking. This constraint does not prevent a component from sending any number of messages, but it entails that the Moderator can only process a finite number of these messages simultaneously. It also implies that we do not consider protocols that could coordinate simultaneously an unbounded number of components.

With regard to the participants in a conversation coordinated by a Moderator, we assume that they have a sequential behavior. In particular, this holds for the Vendor and Customer roles of our example (see Figure 1) since their PN is a state machine (if the communication places are removed, each transition of the net has a single input place and a single output place). Thus, the behavior of a component playing this role can be described by a deterministic automaton whose transitions are labelled either by an event of the kind *!message*, corresponding to the sending of a message, or by an event of the kind *?message*, corresponding to the reception of a message. Figures 1(a) and 1(c) show the automata corresponding to the Vendor and to the Customer roles.

For each role R, an automaton TS_R can be built automatically from the IPN of the Moderator, in order to describe all the sequences of events that a participant playing that role can execute in the course of a conversation of the protocol [18].

In order to relate the Petri net of a moderator to the transition systems of its roles, we introduce some definitions.

Definition 2.1 A labelled Petri net is a structure $N = (P, T, Pre, Post, l)$ where:

- P is the finite set of places and T is the finite set of transitions.
- $Pre : P \times T \to \{0, 1\}$ and $Post : T \times P \to \{0, 1\}$ are the incidence functions defining the arcs from places to transitions and from transitions to places.
- $l : T \to E$ is the labelling function defining the event that is performed at an occurrence of a transition. The label of a transition sending or receiving a message m_1 is respectively $!m_1$ and $?m_1$, while the other transitions are labelled by the null event τ . A sequence of transitions $s = t_1.t_2 \ldots t_n \in T^*$ is labelled as $l(s) = l(t_1).l(t_2) \ldots l(t_n) \in E^*$.
- A marking is a function $M : P \to \mathbb{N}$ that defines the number of tokens staying in each place of the net.

- A transition $t \in T$ may occur, or is enabled, under a marking M if and only if: $\forall p \in P, Pre(p,t) \leq M(p)$.
- The occurrence of a transition that is enabled under a marking M yields a new marking M' such that $\forall p \in P, M'(p) = M(p) - Pre(p,t) + Post(t,p)$. A sequence of transitions $s = t_1.t_2 \ldots t_n$ may occur under a marking M_1 if there exists a sequence of markings $M_2, \ldots, M_{n+1}$ such that each t_i may occur under M_i and its occurrence yields M_{i+1}, for $i = 1 \ldots n$.
- For a marking M of a net N, the language of N from M is the set $\mathcal{L}(N,M) = \{l(s) : s \text{ may occur under } M\}$. When there is no ambiguity on the initial marking M_{init} of the net, we note $\mathcal{L}(N)$ instead of $\mathcal{L}(N, M_{init})$.

Definition 2.2 A labelled transition system is a structure $A = (Q, T, q_0, F, l)$ where:

- Q is the set of states.
- $T \subseteq Q \times Q$ is the set of transitions.
- $l : T \rightarrow E$ is the labelling function defining the event that is performed at an occurrence of a transition.
- $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of its final states.
- A *path* from state q_1 is a sequence of states $p = q_1.q_2 \ldots q_n$ such that $(q_i, q_{i+1}) \in T$. The label of such a path, or the set of events realized by this path, is the sequence $l(p) = l((q_1,q_2)).l((q_2,q_3)) \ldots l((q_{n-1},q_n))$, and the *language* of A from a state q is the set $\mathcal{L}(A,q) = \{l(p) : p \text{ is a path from } q\}$.
- A path of A is a path from the initial state q_0, and $\mathcal{L}(A) = \mathcal{L}(A, q_0)$.

For any participant C in a conversation and a sequence of transitions s of the Moderator's IPN that may occur during a conversation, $s|_C$ denotes the subsequence of s that contains the transitions which exchange messages with C: $s|_C$ is the part of the Moderator's activity that interacts with C during the part s of a conversation. Besides, $(s|_C)^{-1}$ denotes the dual viewpoint on the sequence s: $(s|_C)^{-1}$ denotes the sequence of events that C performs to send (resp. receive) the messages that are expected (resp. sent) by $s|_C$.

Properties

Let IPN be the internal Petri net of the Moderator of a protocol, TS_R be the labelled transition system of a role R, and C be a participant performing role R. The following properties relate the Moderator's IPN to the automata of roles :

(i) TS_R is a finite state deterministic labelled transition system.

(ii) For any sequence of transitions s of IPN enabled at IPN's initial marking, there exists in TS_R a path from the initial state which implements $(s|_C)^{-1}$:

$$\forall s \in \mathcal{L}(IPN) : (s|_C)^{-1} \in \mathcal{L}(TS_R)$$

(iii) Conversely, for any role R, and any path s_R in TS_R starting from its initial state, there exists a sequence s of transitions of IPN enabled at IPN's initial marking, such that the sequence of messages sent and received by s_R is exactly the sequence of messages received and sent by s:

$$\forall s_R \in \mathcal{L}(TS_R) \ : \ \exists s \in \mathcal{L}(IPN) \ : \ s_R = (s|_C)^{-1}$$

(iv) IPN is in its final state iff TS_R is also in its final state.

3 Adaptation approach

An adaption is a change in the coordination of the components that take part in a conversation. We consider adaptations that do not modify the *signature* of the protocol, that is the type of the messages that are exchanged between the Moderator of a conversation and the components. We define an adaptation of a protocol as an unspecified change in the state of the Moderator of a conversation such that, after an occurrence of this adaptation, the set of the sequences of messages that the Moderator can send and receive can be different from before this occurrence. Now, the question arises whether the conversation can reach its final state without changing the behavior of the components.

Inasmuch as the use of coordination rules involves a contractual aspect binding the participants, one can expect that participants agree on an adaptation of these rules. However, in this paper we will not discuss how the participants can reach a consensus on the relevance of an adaptation, or, on the contrary, how some participants can introduce adaptations without the other participants knowing it.

Firstly, we define adaptations more precisely, in this setting. Then, we consider the properties that adaptations should meet in order to keep the conversation consistent with respect to the initial coordination rules. Lastly, we outline how these dynamic adaptations can be managed and implemented.

3.1 Definition of an adaptation

We define an *adaptation* as a change in the Moderator's state that does not comply with the coordination rules. This change is implemented as a new *transition*, which is added to the Moderator's Internal Petri Net and realizes the state change when it occurs, together with the inhibition of a number (possibly null) of transitions in IPN. Thus, the adaption may occur from any marking of the net under which this transition is enabled, and the state change is the one resulting from the transition occurrence. We require that, after any occurrence of the adaptation transition, IPN is bounded and has the proper terminaison property, i.e. its final state is always reachable. Since IPN is bounded from its initial marking, there is a finite number of cases to consider and these properties can be decided.

For example, Figure 2 shows an adaptation of the online auction web site: a transition (T_A) has been added to the Moderator, so that the Vendor receives a payment only after the receipt of the goods has been acknowledged[4].

When needed, we will distinguish the *occurrence* of an adaptation from the *type* of an adaptation. The former denotes the execution of the state change correspond-

[4] Moreover, $T2$ must be inhibited in the adapted Moderator, in order to make the adaptation effective.

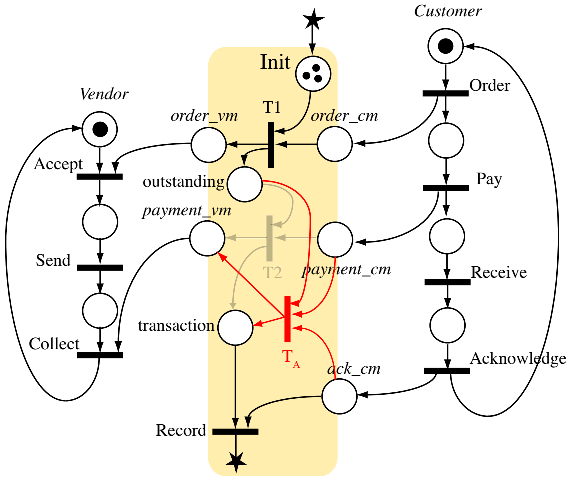


Fig. 2. Adapted Online auction website

ing to the adaptation, from a given point throughout the course of a conversation, while the latter is characterized by the new transition.

3.2 Transparency of an adaptation with respect to a role

Each participant in a conversation holds some role. This role defines the sequences of messages that it can send to and receive from the Moderator. From the viewpoint of a participant in a conversation, an adaptation is *transparent* if the participant cannot detect the occurrence of this adaptation. In fact, the sequence of messages that a participant receives after the adaptation occurrence is one of the sequences of messages it may have received, if the adaptation had not occurred.

But this condition requires that the Moderator is able, from its new state, to receive any message that can be sent by the participant. This leads to the following definition of transparency:

Definition 3.1 [Transparency] Let $\mathcal{A}$ be an adaptation of a coordination protocol $\mathcal{P}$, R be a role of $\mathcal{P}$, *IPN* be the internal Petri net of $\mathcal{P}$, and t be the transition that implements $\mathcal{A}$ in *IPN*. Adaptation $\mathcal{A}$ is *transparent with respect to role* R if and only if, for any marking M of *IPN* reached just after an occurrence of t, and any possible state q of a participant C performing role R when IPN is in state M:

(i) For any sequence of transitions s of IPN enabled at M, there exists in TS_R a path from q which implements $(s|_C)^{-1}$:

$$\forall s \in \mathcal{L}(IPN, M) : (s|_C)^{-1} \in \mathcal{L}(TS_R, q)$$

(ii) Conversely, for any path s_R in TS_R starting from q, there exists a sequence s of transitions of IPN enabled at M, such that the sequence of messages sent and received by s_R is exactly the sequence of messages received and sent by s:

$$\forall s_R \in \mathcal{L}(TS_R, q), \exists s \in \mathcal{L}(IPN, M) : s_R = (s|_C)^{-1}$$

The adaptation in Figure 2 is transparent, as it changes neither the sequence of messages received by the Moderator from the Customer, nor the sequences sent by the Moderator to the Vendor, provided the Customer and the Vendor behave accordingly to their roles.

Since IPN is bounded from its initial marking, there is a finite number of markings M to consider, and TS_R has a finite number of states, too. $\mathcal{L}(IPN, M)$ is a regular language since IPN is bounded from marking M and thus transparency is a decidable property.

3.3 Properties of a safe adaptation

An adaptation of a coordination protocol is *safe* if it does not prevent the Moderator from coordinating the participants although each one keeps the behavior of its role: The Moderator always sends to each participant a sequence of messages that belong to its role definition, and it is able to process the received messages. Even if, due to an adaptation, a conversation departs, at some point, from the coordination rules, this "adapted" conversation is still always able to reach its final state.

Definition 3.2 [Safety] An adaptation of a protocol is *safe* iff: after any occurrence in the course of a conversation following this protocol, if the behavior of each participant in the conversation follows the coordination rules of its role, then the final state of the conversation can always be reached.

Thus, the safety of an adaptation comprises, on the one hand, a constraint on the Moderator's resulting state, which must allow to reach the conversation's final state, and, on the other hand, a constraint on the sequences of interactions between the Moderator and the participants, which must be able to take part in the conversation consistently with respect to their role. In particular, an adaptation should not lead a conversation into a state where either the Moderator is blocked, and thus cannot synchronize the participants anymore, or one of the participants is blocked because it does not receive a message it requires in order to proceed according to its role. Such a blocking of a conversation occurs when the current state of the Moderator or of one the participants is such that it neither receives a message that it needs for going on, nor is able to send a message which is expected.

The notion of safety can be related to the more general concept of conformance (i.e. substitutability with respect to simulation and stuck-freeness) in labelled transition systems [15] in the following sense: an adaptation of a coordination protocol is safe if the whole adapted system (i.e. the adapted Moderator plus the participants) conforms to the non-adapted system. In other words, safety appears as a special case of conformance, aiming at characterizing conformant adaptations that are local to a specfiic component, namely the Moderator.

Theorem 3.3 (Safe adaptation) *Let $\mathcal{A}$ be an adaptation of a coordination protocol $\mathcal{P}$. If $\mathcal{A}$ is transparent for each of its role, then it is a safe adaptation.*

Proof. We have to show that, except at the final state, it never occurs that the Moderator and all components C can neither send nor receive a message. Let us remark that, if the adaptation is transparent for the role of C, properties i) and ii) of Definition 3.1 hold for any marking of IPN and any state of C that are reachable after the occurrence of the adaptation. Thus we only have to show that there is no deadlock after the occurrence of the adaptation. Let M be the marking of IPN and q be the state of C. By definition of an adaptation, $\mathcal{L}(IPN, M)$ is not empty. If it contains a sequence that begins with a send action, the transition labelled by this action may occur in IPN. Else, each sequence in $\mathcal{L}(IPN, M)$ begins with a receive action and, from i) of Definition 3.1, C is able to send the corresponding messages. From ii) of Definition 3.1 we can assert that C cannot send a message that IPN is not expecting and thus cannot process. The reverse reasoning shows that, if C is blocked waiting for messages from IPN, it will necessarily receive one of these messages. Thus, when IPN effectively interacts with the coordinated components, the resulting system is deadlock-free. It remains to show that the system has no livelock, that is if IPN and the components enter a loop, they always have the possibility of exiting. If IPN is in a loop having a send message exit, it can fire this transition. Otherwise, if all exits are receive actions, then from i) of Definition 3.1, there always exists a participant that can send an expected message. □

3.4 Performing adaptations

Adaptations can be used in two different settings:

- either offline: the Moderator's designer is in charge of providing a set of adaptation schemes available to future users. The original design thus includes the initial coordination and its adaptations, which can be activated on request. In this case, the safety of these adaptations can (and should) be verified before execution on the grounds of Theorem 3.3;
- or at runtime: then, the safety property of a requested adaptation should be verified on demand by a specific adaptation service. Safe adaptations can then be registered and activated as in the previous case. In this case, checking the transparency conditions (Definition 3.1) is simpler than in the previous case, as we only have to consider the Moderator's current marking, M, and the current state q of each participant.

Safe adaptations can be activated by the Moderator either temporarily or permanently. In the former case, the adaptation will only have one occurrence, whereas in the latter case, it will remain enabled until the user explicitly requests its removal, or until the current conversation reaches an end.

To carry out a requested safe adaptation, we can investigate two different implementations :

- either the Moderator has a monitoring service, which blocks the transitions inhibited by the adaptation, performs the adaptation state change as soon as the condition for its occurrence is true, and then unblocks the inhibited transitions.
- or, the Moderator's PN already contains the transition t which implements a given (predefined) adaptation $\mathcal{A}$. Then, the activation of $\mathcal{A}$ will be implemented

by temporarily setting t's priority from a negative level (i.e. t is not enabled) to the highest level (i.e. t will be fired before any other enabled transition), and conversely from a positive to a negative priority level for the inhibited transitions.

4 Case study

We illustrate our approach with a case study about the control of accesses to documents by students during an online examination. Figure 3 describes the interactions between the e-learning computer system (ECS), the assistant and any student.

To participate to the examination, a student has first to log in to the ECS. According to the student's profile, the ECS determines the list of documents that this student is granted to access. This list of authorized documents is supplied to each student at the beginning of the examination. However, purposely or not, a student may access a non-authorized document.

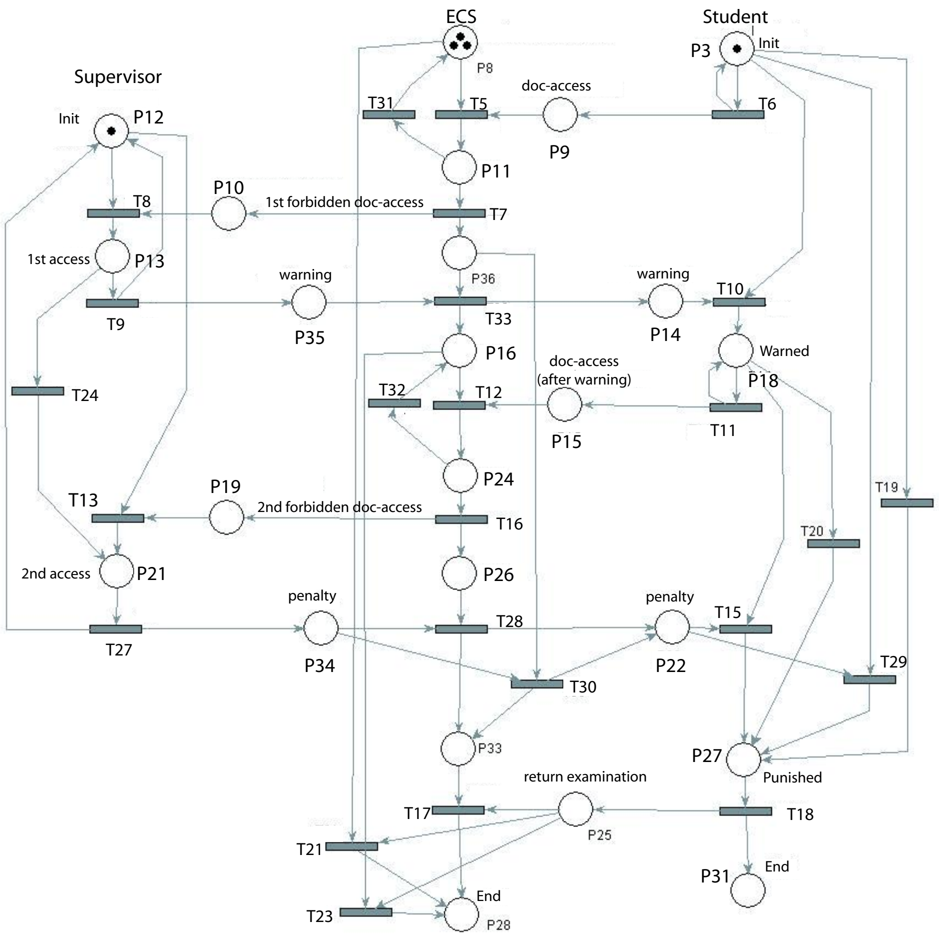


Fig. 3. Global description of the cooperation related to the control of accesses to documents

The e-learning computer system (ECS) acts as the Moderator of the examination process, where a single participant plays the Supervisor role while the Student role may be played by several participants.

The ECS executes the PN that is the central part of figure 3. At the beginning of an examination, the initial place P8 contains one colored token for each student, and so does the terminal place P28 at the end of the examination. The left-hand part of Figure 3 shows the behavior of the supervisor role, while the right-hand part shows the behavior of any student.

Access document checking is performed by the ECS. To this end, the workstation of each student informs the ECS of any document access (transition $T6$, or transition $T11$ if the student has already received a warning). The ECS then checks whether this document belongs to the list of documents authorized for the student (transition $T5$, or $T12$). If the document is authorized, transition $T31$ (or $T32$) occurs. If it is not the case, the system informs the assistant that supervises the examination (transition $T7$, or $T16$).

A faulty access may be handled in three ways:

(i) The access violation is the first, but is a major one. The assistant immediately imposes a penalty to the student (sequence of transitions $T8 \rightarrow T24 \rightarrow T27$).

(ii) It is the student's first unauthorized access, and this access is not serious. The assistant warns the student not to repeat this action (sequence $T8 \rightarrow T9$).

(iii) It is the second unauthorized access of the student. Regardless of the access seriousness, the student is imposed a penalty (sequence $T16 \rightarrow T13 \rightarrow T27$).

When the student has to bear a penalty, the penalty is recorded by the ECS (transitions $T27$ and then $T28$, or $T30$), and the student must stop the examination (transition $T29$ if it is after the first unauthorised access, or $T15$ if it is after the second one) and send his work to the computer system (transition $T18$). Transitions $T17$, $T23$ and $T21$ record the student's work in the various cases.

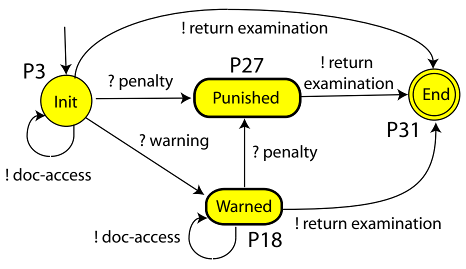
4.1 Access control policy for documents

Figure 4(a) gives the student behavior in which transitions are either receiving or sending. The current state of the automaton always remains its initial state while accesses are allowed. A forbidden access leads to either a warning or a penalty.

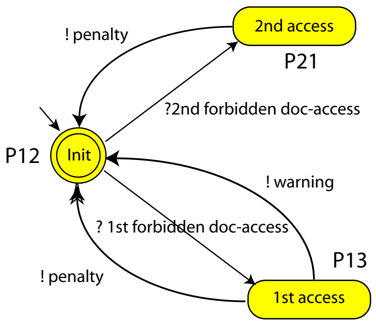
Figure 4(b) describes the supervisor behavior. From its initial state, forbidden accesses are signaled and lead to either warning or penalty messages.

4.2 Adaptation example

The current Moderator reports unauthorized accesses to the assistant. The first time, the assistant can choose either to warn the student or to assign a penalty (see place $P13$ and transitions $T9$ or $T24$ in figure 3). The second time, the assistant assigns a penalty (see transition $T13$). A more rigorous or a more tolerant attitude can be implemented by means of an adaptation.



(a) Student automaton



(b) Supervisor automaton

Fig. 4. Automaton graphs

A more rigorous adaptation:

Figure 5(a) illustrates a more rigorous attitude which consists in assigning a penalty as soon as the first forbidden access occurs. In this case, when a token enters $P11$, a new transition, $T35$ is introduced. This transition deletes this token and puts a token into $P24$. With respect to the Moderator, the document access (*doc-access* place) is now directly interpreted as a document access after warning (*doc-access (after warning)* place). This new behavior of the ECS remains consistent with the student behavior and the requesting role, namely the assistant, behavior. In this case, the Moderator must be prevented from signalling the first forbidden access to the Supervisor. As stated in Section 3.4, this can be implemented either by giving the adaptation transition ($T35$) a higher priority than transition $T7$, or by inhibiting transition $T7$.

This adaptation is safe because, according to Theorem 3.3, this adaptation is transparent for the supervisor and student roles: the accessible states of the supervisor automata become $(Init, 2nd\ access)$ and, for the student automata, the direct path $(Init \rightarrow Punished \rightarrow End)$ remains in use.

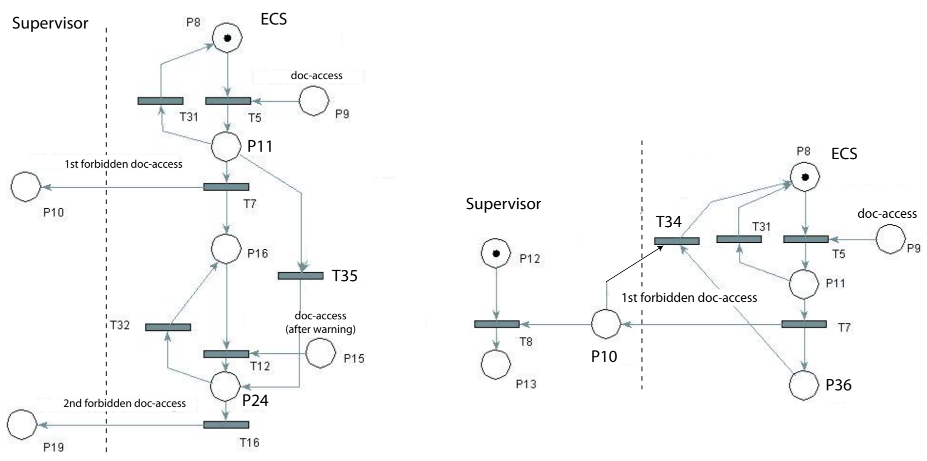
A more tolerant adaptation:

Figure 5(b) illustrates a more tolerant attitude which consists in hiding alert occurrences to the Supervisor. In such a case, when tokens are present in $P10$ and $P36$, a new transition, $T34$ deletes these tokens and puts a new token into $P8$. Thus, the PN state returns to a previous marking.

Such an adaptation again is safe because of the transparency property with respect to the roles. In this case, accessible states of the student automata become $(Init, 1st\ access)$ and the supervisor automata remains in the initial state.

A counter-example:

The assistant could allow a student to make more than two unauthorized accesses. This adaptation would be implemented by a transition moving a token from $P15$ (*doc-access (after warning)*) toward $P9$ (*doc-access*). But this adaptation does not satisfy our safety criteria. Indeed, the arrival of the token in $P9$ can cause a token to be put into $P14$ (*warning*), while the student is no longer able to process this token: transition $T10$ is not enabled because the student's token stays in $P18$ instead of $P3$. In this state, no transition may occur in the student's net.



(a) A more rigourous attitude (b) A more tolerant attitude

Fig. 5. Local adaptation of the PN

5 Conclusion

We propose a framework for specifying the coordination between components, handling and checking adaptations of this coordination. In particular, we define a computable safety criterion, which allows to determine automatically whether an adaptation is correct. Moreover, we propose a pattern for checking and handling (dynamic) adaptation requests, that comes into the more general framework proposed in [14].

Putting our proposal into practice undergoes taking into account limits and issues which are inherent to the field of operational models, namely:

- The verification of the safety of an adaptation involves a cost insofar as a validation must be run onto the PN. This validation is grounded on an exhaustive system state space exploration, and its limits, in general, are those of model checking. The validation of the safety of an adaptation can though benefit from works on conformance checking [15]. This can lead to constrain the PN modifications so that this analysis remains scalable according to the PN size.
- The lack of modularity, which does not enable to design adaptations at a higher level of abstraction than the original transition system. On practical grounds, our approach is sensible, as we only consider the validation of adaptations de-

fined by programmers, but it is less attractive than the approach that consists in generating a set of correct adaptations.

The idea of safety (of an adaptation) addresses both complexity and modularity as it aims at characterizing and verifying the impact of a localized adaptation in a component (namely, the Moderator) of a composite system. In this spirit, our approach could be prolonged in several ways:

- By defining a stronger safety criterion to characterize safe adaptations, independently from roles and thus from transparency. This criterion, based in the substitutability between the initial Moderator, and an adapted Moderator would enable to check locally a local adaptation.
- Transparency could be used to check that an adaptation (an arbitry change of state) of a participant remains consistent with an existing coordination protocol: if the Moderator's IPN is transparent with respect to the adapted role, then the new (global) system conforms to the initial system. Again, we have a means to check locally a local adaptation.

If one wants to implement complex adaptations, the composition of adaptations is another important issue: an adaptation can be safe from the initial definition of the coordination rules, and become unsafe after having performed another adaptation. Therefore, the safety must be checked by comparing the "adapted language" of the Moderator (that is the language resulting from the coordination plus the already integrated adaptations) with the language resulting from the additional adaptation.

From a software engineering perspective, our approach must be supported by a set of tools, in order to facilitate the checking of the correctness of the design and of the adaptation of coordination protocols:

- We provide a tool generating the role automata from the Moderator's IPN [18].
- For usability purposes, the results of the safety analysis should be made available to the programmer, in order to allow the debugging of adaptation requests.
- More generally, the programmer should be provided with tools enabling him to assess the impact of an adaptation on overall performance [3].

Lastly, this proposal opens the way to a methodological approach; the designer can specify a basic scenario which ensures a core of safety properties and consequently a rich set of behaviours. Then, specific rules can be introduced in an incremental way, as adaptations. This could provide a smooth and flexible design process.

Acknowledgement

We wish to thank the anonymous referees for their constructive comments which significantly helped to improve the content of the paper. Moreover, we gratefully acknowledge the help of Pascal Poizat during the last revision phase of this paper.

References

[1] Allen, R. and D. Garlan, *A Formal Basis for Architectural Connection*, ACM Transactions on Software Engineering and Methodology **6** (1997), pp. 213–249.

[2] Baldoni, M., C. Baroglio, A. Martelli, V. Patti and C. Schifanella, *Verifying the Conformance of Web Services to Global Interaction Protocols : a First Step*, in: M. Bravetti and G. Zavattaro, editors, *European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005*, LNCS **3670** (2005), pp. 257–271.

[3] Becker, S. and R. H. Reussner, *The Impact of Software Component Adaptors on Quality of Service Properties*, in: C. Canal, J.-M. Murillo and P. Poizat, editors, *First International Workshop on Corordination and Adaptation Techniques for Software Entities (WCAT'04)*, L'objet **12** (2006), pp. 105–125.

[4] Burdett, D. and N. Kavantzas, *WS Choreography Model Overview*, Working Draft, W3C (2004).
URL http://www.w3.org/TR/ws-chor-model/

[5] Canal, C., J. M. Murillo and P. Poizat, *Software Adaptation*, L'Objet **12** (2006), pp. 9–31, guest Editor's Introduction to "Special Issue on Coordination and Adaptation Techniques for Software Entities".
URL http://www.ibisc.univ-evry.fr/pub/basilic/OUT/Publications/2006/CMP06

[6] Canal, C., P. Poizat and G. Salaün, *Synchronizing Behavioural Mismatch in Software Composition*, in: *Eighth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, LNCS **4037** (2006), pp. 63–77.

[7] Castelfranchi, C., *Engineering Social Order*, in: A. A. Omicini, R. Tolksdorf and F. Zambonelli, editors, *First International Workshop on Engineering Societies in the Agents World (ESAW 2000)*, LNCS **1972** (2000), pp. 1–18.

[8] de Alfaro, L. and T. A. Henzinger, *Interface Automata*, in: *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, ACM Press, 2001, pp. 109–120.
URL http://www.gigascale.org/pubs/239.html

[9] Gorton, I., G. T. Heineman, I. Crnkonic, H. W. Schmidt and J. A. Stafford, "Component-based Software Engineering," LNCS **4063**, Springer Verlag, 2006.

[10] Hanachi, C. and C. Sibertin-Blanc, *Protocol Moderators as Active Middle-Agents in Multi-Agent Systems*, Autonomous Agents and Multi-Agent Systems **8** (2004), pp. 131–164.

[11] Heineman, G. T. and W. T. Councill, "Component-based Software Engineering: putting the Pieces together," Addison-Wesley Professional, 2001.

[12] Issarny, V., C. Kloukinas and A. Zarras, *Systematic Aid for Developing Middleware Architectures*, ACM Communications **45** (2002), pp. 53–58.

[13] Murata, T., *Petri Nets: Properties, Analysis and Applications*, Proceedings ot the IEEE **77** (1989), pp. 541–580.

[14] Poizat, P., G. Salaün and M. Tivoli, *On Dynamic Reconfiguration of Behavioral Adaptations*, in: S. Becker, C. Carlos, N. Diakov, J. M. Murillo, P. Poizat and M. Tivoli, editors, *Third International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'06)*, 2006, pp. 61–69.

[15] Rajamani, Sriram K. and Rehof, Jakob, *Conformance Checking for Models of Asynchronous Message Passing Software.*, in: *Proceedings of the 14th International Conference on Computer Aided Verification*, LNCS **2404** (2002), pp. 166–179.

[16] Shaw, M., R. DeLine and G. Zelesnik, *Abstractions and Implementations for Architectural Connections*, in: *Third International Conference on Configurable Distributed Systems*, Annapolis, Maryland, 1996, pp. 2–10.
URL http://www-2.cs.cmu.edu/~Vit/paper_abstracts/ArchCxn.html

[17] Sibertin-Blanc, C., *Cooperative Objects: Principles, Use and Implementation*, in: G. Agha and F. De Cindio, editors, *Petri Nets and Object Orientation*, LNCS **2001** (2000), pp. 210–241.

[18] Sibertin-Blanc, C. and N. Hameurlain, *Participation Components for Holding Roles in MultiAgents Systems Protocols*, in: *ESAW'04,Engineering Societies in the Agents World*, LNAI **3451** (2004), pp. 60–73.

[19] Tivoli, M., P. Fradet, A. Girault and G. Goessler, *Adaptor Synthesis for Real-Time Components*, in: O. Grumberg and M. Huth, editors, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007)*, LNCS **4424** (2007), pp. 185–200.

[20] Yellin, D. M. and R. E. Strom, *Protocol Specifications and Component Adaptors*, ACM Transactions on Programming Languages and Systems (TOPLAS) **19** (1997), pp. 292–333.