Pattern-Based Modeling and Formalizing of Business Process Quality Constraints

Lial Khaluf, Christian Gerth, and Gregor Engels

Department of Computer Science
University of Paderborn
Paderborn, Germany
lial.khaluf@googlemail.com,
{gerth,engels}@uni-paderborn.de

Abstract. The quality of business processes can be checked by verifying their compliance with specific quality constraints. These constraints represent a set of required temporal and logical relationships between different steps of business processes. Quality constraints are usually formulated as informal texts, which makes them difficult to be verified, when business processes become complex. One way to solve this problem is by automating the verification of quality constraints on business processes by applying model checking. To apply model checking, both business processes and quality constraints have to be formalized. In this paper, we define a new visual language for modeling quality constraints and we provide a pattern-based translation for quality constraint models into Computation Tree Logic formulas.

Keywords: business process, quality constraint, visual pattern, CTL-formula.

1 Introduction

One of the most important factors of the success and reputation of any business is the quality of products and services it provides. For this reason, quality management has become an important competitive factor that must be considered on all levels including business processes. In this context, many standards were developed for total quality management, aiming at fulfilling the requirements of customers and improving the quality of products, as e.g. ISO 9001 regulations and constraints [1], which can be applied to any business. Quality constraints may be defined by producers or by customers. No matter who defines quality constraints, there must be a way to ensure that business processes satisfy them. However, since standard or user-defined quality constraints are usually documented as informal texts, it becomes difficult to prove their correctness, especially when business processes are complex. One solution for this problem is automating the verification of quality constraints on business processes by using the technique of model checking. To apply this technique, both business processes and quality constraints have to be formalized. To achieve this goal, many approaches were developed, where each one depends on a different temporal logic to formalize quality constraints, in order to enhance and increase their expressiveness. However, the major problem is still that no approach allows to formalize user-defined quality

constraints, which include a non-deterministic future, as e.g. formalizing a quality constraint or a part of it which must not hold for all control flows in a business process, but for at least one control flow.
Our goal in this paper is to overcome the expressiveness limitations concerning the branching logic, by allowing to formalize non-deterministic user-defined quality constraints, and to make this formalization intuitively understandable for business process users. For this reason, we have investigated the formalization approaches, which are based on UML Activity Diagrams [3], since these diagrams are a widely used standard and are familiar to business process users. One of these approaches is developed in [2]. It models business processes as UML 2.0 Activity Diagrams, and quality constraints as business process patterns using the Process Pattern Specification Language (PPSL) [4], which is a light weight extension of UML Activity Diagrams. Both business processes and business process patterns are then written in a formal way by transforming UML 2.0 Activity Diagrams into labeled transition systems, and translating business process patterns into LTL-formulas [5]. This enables the automated verification of quality constraints on business processes by using a model checker to verify LTL formulas on labeled transition systems. However, the Linear Temporal Logic (LTL) does not support the non-deterministic future. To support this kind of future, we replace LTL by the Computation Tree Logic (CTL) [6]. To achieve that, we extend PPSL to a new visual modeling language, Extended Process Pattern Specification Language (EPPSL), which has a formally defined semantics given by a translation into CTL-formulas. EPPSL is a heavy weight extension of UML Activity Diagrams. In other words, EPPSL uses the elements of UML Activity Diagrams, which semantics can serve to model quality constraints. It also defines new classes of elements to cover the semantics of quality constraints which are not defined by UML Activity Diagrams. We also provide a pattern-based translation for EPPSL models into CTL-formulas. In the following section, we give an overview of the related work. In Section 3, we provide a scenario for verifying business process quality constraints. In Section 4, we describe the modeling elements of EPPSL and how to use them for composing EPPSL models. In Section 5, we explain how to translate EPPSL models into CTL-formulas. In Section 6, we provide a conclusion and outlook for our approach.

2 Related Work

Modeling and formalizing business process constraints in order to verify their correctness are considered in several approaches. For example, PPSL is introduced in [4] and translated in [2] into LTL formulas. In [7], the graphical Business Property Specification Language is used to capture business process compliance rules, which are translated into LTL. In [8], DecSerFlow is mapped on LTL. DECLARE is defined in [9] and [10]. DECLARE models are also translated into LTL. BPMN-Q is developed in [11] to model requested compliance rules on business processes as queries. These queries are translated in [12] into PLTL [13][14] formulas. BPMN is used in [15] to check the semantic correctness of business process models by mapping them to Petri nets. In [16], the Object Constraint Language [17] expressions are used to refer to an integrated metamodel for different process models. In [18], process-independent compliance rules are specified using graph structures and formalized in terms of FOL [19]. All the previous approaches are not sufficient to express constraints, which contain a non-deterministic future and which have the degree of complexity required by users. But, in our approach, we define EPPSL which allows the users to build high complex shapes of quality constraints and we support the non-deterministic future by formalizing quality constraints as CTL-formulas. In [20], a lightweight, analyst-mediated approach introduces compliance patterns in terms of CTL as a heuristic basis for resolving the non compliance of process models, but it does not provide a way to map informal user-defined compliance rules into CTL-formulas. Additionally, expressing the non-deterministic future is limited to a set of structural and semantic patterns, which have a predefined shape of rules, but in our approach, we define EPPSL to enable modeling and formalizing quality constraints which can reach the complexity of CTL expressions.

3 Scenario

We assume that a company wants to hire new employees. The company has developed a business process to model the activities that must be carried out to accept or refuse applications. Fig. 1 shows the business process modeled as a UML Activity Diagram. To ensure the quality of this business process, some quality constraints may have to be checked on it, e.g. if a business analyst wants to know whether the lack of employees could result in accepting the application, then the correctness of QC_1 must be checked (We use an abbreviation QC to refer to a quality constraint).

QC_1: *"It is always the case, that when the number of required employees has not yet been reached, then there exists a possibility to accept the application".*

By looking at the business process model, we see that it satisfies QC_1, since whenever we encounter the Guard "the number of required employees has not yet been reached", there exists at least one control flow, on which this Guard is followed by the Action "accept the application" where in between other Actions may occur and other Guards maybe be satisfied. However, verifying a textual quality constraint by looking at the business process model is prone to errors, especially when business processes become complex. In order to be able to verify QC_1 automatically on the business process model in Fig. 1 , we have to model QC_1 using specific patterns, which have a formally defined semantics given by a translation into temporal logic formulas.

The expression *"there exists a possibility"* in QC_1 states that the lack of employees enables the possibility of accepting the application, but does not enforce it. In other words, QC_1 states that accepting the application could be a non-deterministic future for the lack of employees. In order to have the ability to express this kind of future, it is not enough that we model QC_1 using specific patterns, which have a formally defined semantics given by a translation into temporal logic formulas, but also the temporal logic formulas should have the ability to express a non-deterministic future.
In this paper, we define a new modeling language EPPSL, which provides the patterns required by the previous scenario, since these patterns have a formally defined semantics given by a translation into CTL-formulas. We specify EPPSL by a meta model which we provide in [21]. In the following section, we introduce how to model quality constraints with EPPSL.

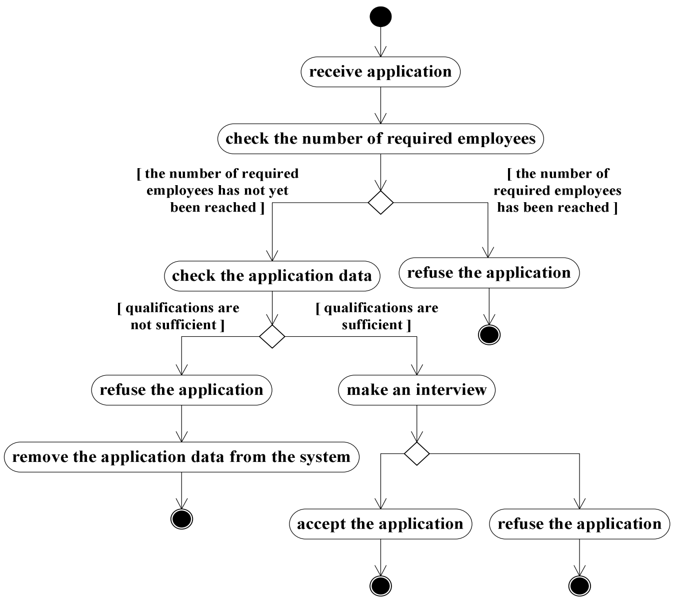

receive application
check the number of required employees
[the number of required employees has not yet been reached]
[the number of required employees has been reached]
check the application data
refuse the application
[qualifications are not sufficient]
[qualifications are sufficient]
refuse the application
make an interview
remove the application data from the system
accept the application
refuse the application

Fig. 1. Example of a business process model

4 Modeling Quality Constraints with EPPSL

We model a quality constraint by modeling its basic blocks and its temporal and logical relationships. The basic blocks could be actions, guards, anonymous steps, and partial quality constraints. The temporal and logical relationships are based on the semantics of the temporal and logical relationships of CTL [6] to provide the ability to express both deterministic and non-deterministic futures. In the following, we explain the semantics of both basic blocks and relationships and how to model them.

4.1 Modeling Basic Blocks

A quality constraint may include a mixture of the following basic blocks:

Actions: An action is an activity which is carried out by the system, the customer or any other entity in the business process. For example, QC_1 consists of the action "accept the application". The start of a business process and the end of a control flow in a business process are also considered to be actions. EPPSL uses the modeling elements "Action" (Fig. 2.a), "InitialNode" (Fig. 2.b), and "ActivityFinalNode" (Fig. 2.c) to model actions, the start of a business process, and the end of a control flow in a business process, respectively.

Guards: A guard is a condition which can be false or true. For example, QC_1 consists of the Guard "the number of required employees has not yet been reached". EPPSL uses the modeling element "Guard" (Fig. 2.d) to model guards.

Anonymous steps: A quality constraint may refer to an anonymous step which could be an unknown action or an unknown guard. For example, a quality constraint might be dedicated to ensure that the business process model in Fig. 1 includes a possibility to accept the application after 6 steps from starting the process without any need to know to which actions or guards these steps are referring. Quality constraints which counts anonymous steps are useful if the number of steps refers e.g. to the time consumed or the money paid to perform these steps, or if it plays a role in the satisfaction of the customer. For this reason, we introduce in EPPSL a modeling element called "AnonymousStep" (Fig. 2.e).

Partial quality constraints: A partial quality constraint is a quality constraint which is linked to other partial quality constraint(s) with a logical relationship to build another quality constraint. We use the concept of the partial quality constraint, since we need sometimes to model a quality constraint which consists of several quality constraints that are logically related, but temporally not related. For example, a quality constraint might be dedicated to ensure that if the business process model in Fig. 1 includes a possibility to make an interview, then it includes no possibility to accept the application online. The first possibility is not temporally related to the second one. However, they are logically related, since the first possibility implies the negation of the second one. We model partial quality constraints as separated units. For this reason, we provide in EPPSL a modeling element called "ConstraintContainer" (Fig. 2.f) which is dedicated to contain a partial quality constraint model separating it temporally from other partial quality constraint models.

The elements in Fig. 2.a, Fig. 2.b, Fig. 2.c, and Fig. 2.d are the same elements used by UML 2.0 Activity Diagrams to model Actions, InitialNodes, ActivityFinalNodes, and Guards. The elements in Fig. 2.e and Fig. 2.f are new classes defined by EPPSL.

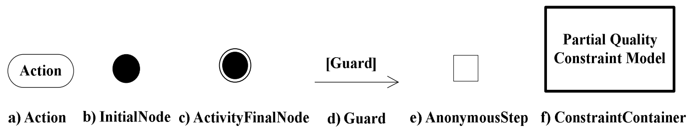


Fig. 2. EPPSL modeling elements for the basic blocks

EPPSL models for quality constraints provide the ability to link the modeling elements of the basic blocks with temporal and logical relationships. In the following, we introduce how EPPSL can model these relationships.

4.2 Modeling Temporal Relationships

A temporal relationship determines the order of actions, guards, and anonymous steps. For example, QC_2, which we want to verify on the business process model in Fig. 1, consists of a temporal relationship "After":

QC_2: "After checking the application data, the qualifications of the applicant are considered to be either sufficient or not sufficient".

The temporal relationship in QC_2 states that the action "check the application data" must be followed by one of the guards "qualifications are not sufficient", or "qualifications are sufficient".

EPPSL provides a set of modeling elements to express temporal relationships. Since EPPSL considers deterministic and non-deterministic futures, it provides for each temporal relationship two modeling elements. The first one represents the relationship when it holds for all control flows of a business process (deterministic). The second one represents the relationship when it holds for at least one control flow (non-deterministic). Fig. 3 shows the notation of EPPSL modeling elements for temporal relationships. These elements are new classes defined by EPPSL.

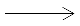
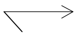
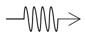
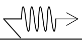
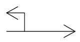
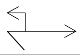
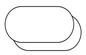
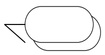
Temporal Relationship	Notation	Temporal Relationship	Notation
Next		PossiblyNext	
After		PossiblyAfter	
Until		PossiblyUntil	
All		PossiblyAll	

Fig. 3. EPPSL modeling elements for temporal relationships

Deterministic Temporal Relationships: The deterministic temporal relationships are "Next", "After", "Until", and "All". "Next" and "After" may link two basic blocks, which could be "Actions", "Guards" or "AnonymousSteps". "Next" states that the first block must be followed next by the second block on all control flows of a business process. "After" states that the first block must be followed by the second block on all control flows of the business process, no matter if other blocks occur between the first and the second block. "Until" connects two basic blocks, which could be two "Actions", or an "Action" and a "Guard". "Until" states that an action must be repeated on all control flows of a business process until another action takes place or a guard is satisfied. "All" refers to all instances of an action on all control flows of a business process. "All" is usually used to confirm that a quality constraint which includes an action, to which the temporal relationship "All" is applied, must hold for all instances of that action on all control flows of a business process.

For example, we want to verify QC_3 on a business process model for using a bank card to withdraw money.

QC_3: "The pin number must always be entered repeatedly until the pin number is correct".

The EPPSL model in Fig. 4 models QC_3. It states that all instances of the action "enter the pin number" must be repeated on all control flows of the business process until the guard "pin number is correct" is satisfied. In this model, we have applied

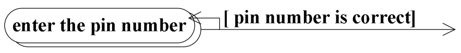


Fig. 4. EPPSL quality constraint model

an "All" temporal relationship on the action "enter the pin number", since the word "always" in QC_3 states that the temporal relationship "Until" must hold for all instances of the action "enter the pin number".

Non-Deterministic Temporal Relationships: The non-deterministic temporal relationships are "PossiblyNext", "PossiblyAfter", "PossiblyUntil", and "PossiblyAll". These relationships have the same semantics of "Next", "After", "Until", and "All" respectively with one different aspect that they must not hold for all control flows of a business process, but for at least one control flow. For example, we want to verify QC_4 on the business process model in Fig. 1.

QC_4: *"There exists a possibility after starting the process to check the application data which is possibly followed by making an interview".*

The EPPSL model in Fig. 5 models QC_4. It states that there exists at least one control flow on which starting the process is followed by checking the application data which is followed on at least one control flow by making an interview.

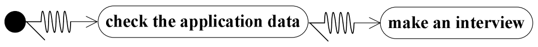


Fig. 5. EPPSL quality constraint model

4.3 Modeling Logical Relationships

A logical relationship may link actions, guards, and partial quality constraints. For example, QC_2 consists of a logical relationship "Or" which combines two guards and states that either the first guard "qualifications are sufficient" or the second one "qualifications are not sufficient" must follow the action "check the application data".
EPPSL provides the following set of modeling elements to express logical relationships, which may link Actions, Guards, and ConstraintContainers:

- **Join/ForkNode:** is used to model the logical relationship "And". "Join/ForkNodes" can be used to link Actions, Guards, and ConstraintContainers. We refer to Join/Fork Nodes as control nodes.
- **Decision/MergeNode:** is used to model the logical relationship "Or". "Decision/ MergeNodes" can be used to link Actions, Guards, and ConstraintContainers. We refer to Decision/MergeNodes as control nodes.
- **Not:** is used to model the "Not" logical operator. "Not" can be applied to Actions, ConstraintContainers, and all EPPSL temporal relationship modeling elements.
- **Connector:** is used to model the "Imply" logical relationship. Connectors can link between two ConstraintContainers, or between ConstraintContainers and control nodes.

Logical Relationship	Notation	Logical Relationship	Notation
Join/ ForkNode	┃	Not	╳
Decision/ MergeNode	◇	Connector	——•

Fig. 6. EPPSL modeling elements for logical relationships

Fig. 6 shows the notation of the EPPSL modeling elements for logical relationships. The "Join/ForkNodes" and "Decision/MergeNodes" are the same control nodes used by UML 2.0 Activity Diagrams. The "Not" and "Connector" elements are new classes defined by EPPSL.

To give an example for using the logical relationships modeling elements, we assume that we want to verify QC_5 on the business process model in Fig. 1.

QC_5: *"If there exists a possibility to make an interview, then there exists no possibility to accept the application online".*

The EPPSL model in Fig. 7 models QC_5. It includes two EPPSL models for two partial quality constraints, which are temporally not related. This means that each one of them must be checked separately on the business process. For this reason, we separate each one in a ConstraintContainer. The first model states that there exists a possibility to make an interview after starting the process. The second model states that there exists no possibility to accept the application online after starting the process. The Constraint-Containers are linked with a Connector, which means that the partial quality constraint represented by the first model must imply the partial quality constraint represented by the second model.

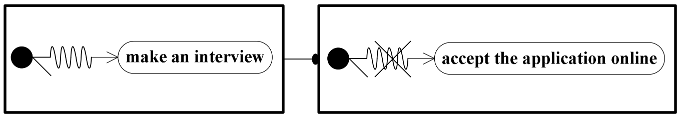


Fig. 7. EPPSL quality constraint model

5 Translation of EPPSL Models into CTL-Formulas

The Computation Tree Logic (CTL) [6] views the time as a tree. It considers all different paths, allowing the future to be non-deterministic. CTL-formulas are based on a set of atomic propositions (statements which truth value may change over time), logical connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$), temporal operators (***X***: Next, ***F***: Eventually, ***U***: Until, ***G***: Globally), and path quantifiers (***A***: On all paths, ***E***: On at least one path). Whenever there is a temporal operator in a CTL-formula, a path quantifier must precede it. For example, if ψ is an atomic proposition, then $\boldsymbol{AX}\psi$ is a CTL-formula, which states that, On all paths, ψ holds next.

In order to formalize quality constraints, we translate their EPPSL models into CTL-formulas. Here, we differentiate between two kinds of models depending on the number of control nodes they contain. Models which contain at most one control node are simple models. Models which contain more than one control node are complex models.

When we translate a simple or a complex model into a CTL-formula, we translate the basic blocks: Actions, Guards, InitialNodes, and ActivityFinalNodes into the atomic propositions: Actions labels, Guards texts, "Start", and "End" respectively. However, the strategy for constructing the CTL-formula for a simple model differs from the strategy for constructing it for a complex model. In the following, we explain the different strategies.

5.1 The Translation Strategy for Simple Models

The translation of simple models depends on analyzing them to specific EPPSL patterns, for which we provide a translation into CTL-formulas [21]. These patterns enable all required combinations of EPPSL modeling elements, in order to cover all possible semantics of quality constraints when the model contains at most one control node. For example, we want to formalize QC_6 which has to be verified on the business process model in Fig. 1:

QC_6: *"It is always true that checking the application data might be followed by refusing the application, which is followed next by removing the application data from the system".*

First, we model QC_6 with EPPSL (see Fig. 8), then we translate the model into a CTL-formula by comparing it to the EPPSL patterns given by the translation tables, which we provide in [21].

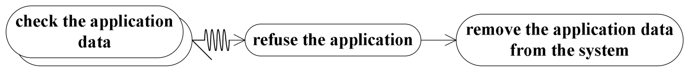

check the application data
refuse the application
remove the application data from the system

Fig. 8. EPPSL quality constraint model

Due to lack of space, we show in Fig. 9 only a part of the translation tables dedicated for simple models, where S represents an EPPSL pattern, which is attached to the element preceding S, and S^* refers to the CTL-formula, to which S is translated. C represents a partial quality constraint model, and C^* refers to the CTL-formula, to which C is translated. M represents an EPPSL modeling element, and M^* refers to the CTL-formula, to which M is translated.

We can translate simple models from right to left or left to right. Here e.g., we translate the model of QC_6 from right to left as explained by Fig. 10. In Step 1, we translate the Action "remove the application data from the system" to an atomic proposition represented by the Action label. In Steps 2, 3, 4, and 5, we divide the model into EPPSL patterns provided by the translation table in Fig. 9. This enables us to translate each pattern separately, until we reach the CTL-formula, which represents the whole model in Step 5.

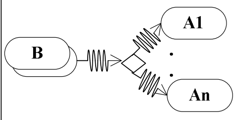
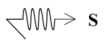
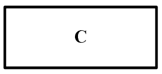
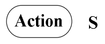
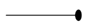
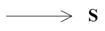
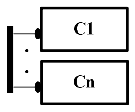
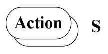
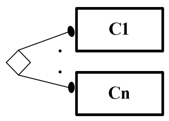
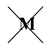
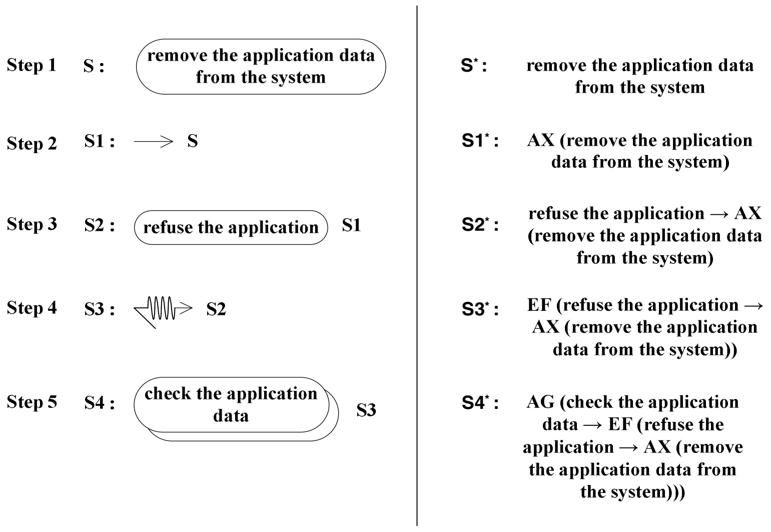
EPPSL Pattern	CTL-formula	EPPSL Pattern	CTL-formula
	AG (B → (AF(A1) V .. V AF(An)))		EF (S^*)
	(C^*)		Action → S^*
	→		AX (S^*)
	($C1^*$) Λ .. Λ (Cn^*)		AG (Action → S^*)
	($C1^*$) V .. V (Cn^*)		¬M^*

Fig. 9. Translation of EPPSL patterns into CTL-formulas



Step 1	S :	remove the application data from the system	S^* :	remove the application data from the system
Step 2	S1 :	→ S	$S1^*$:	AX (remove the application data from the system)
Step 3	S2 :	refuse the application S1	$S2^*$:	refuse the application → AX (remove the application data from the system)
Step 4	S3 :	S2	$S3^*$:	EF (refuse the application → AX (remove the application data from the system))
Step 5	S4 :	check the application data S3	$S4^*$:	AG (check the application data → EF (refuse the application → AX (remove the application data from the system)))

Fig. 10. Example of the translation strategy for a simple EPPSL model into a CTL-formula

5.2 The Translation Strategies for Complex Models

Logical relationships may link actions and guards, or may link partial quality constraints with respect to the basic assumptions defined in [21]. For this purpose, we define two different translation strategies for complex models. The first one is used when more than one control node link Actions and Guards (e.g. the model in Fig. 11), and the second one is used when more than one control node link ConstraintContainers (e.g. the model in Fig. 13).

Strategy 1: Given an EPPSL complex model, e.g. the complex model of QC_7 in Fig. 11, which we want to verify on the business process model in Fig. 1:

QC_7: *"Receiving an application is always followed by making an interview or refusing the application and remove the application data from the system".*

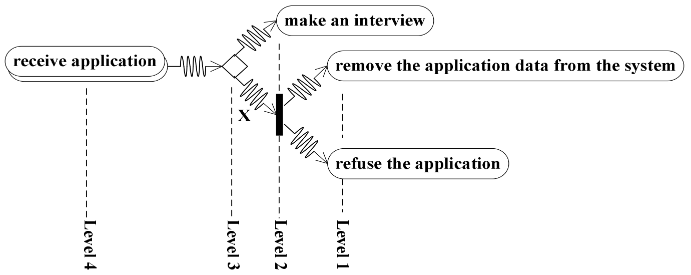


Fig. 11. Example of a complex model

We translate the model in Fig. 11 by applying the following steps:

1. We divide the model into levels numbering them in the opposite direction of the temporal relationships. In Fig. 11, we have 4 levels.
2. We assign a variable name to each temporal relationship followed and preceded directly by a control node. In Fig. 11, we have one variable *X*.
3. When a control node is preceded directly only by a variable with no assigned value and followed directly by temporal relationships attached only to Actions or Guards, then the node has to be translated according to the translation table dedicated for control nodes preceded by variables [21] (due to lack of space, we present only a part of it in Fig. 12) and the resulting CTL-formula is assigned to the variable. According to the table in Fig. 12, we translate the ForkNode on Level 2 in Fig. 11, and we assign its formula to *X*:
   *X: (**AF**(refuse the application)* $\wedge$ ***AF**(remove the application data from the system))*
4. When a control node is followed directly only by a variable with no assigned value and preceded directly by temporal relationships attached only to Actions or Guards, then the node has to be translated according to the translation table of control nodes followed by variables [21], and the resulting CTL-formula is assigned to the variable. Fig. 11 does not encounter this case.

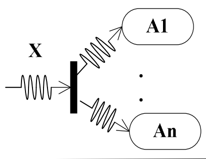
Control Nodes	The Value of X
	(AF (A1) Λ .. Λ AF (An))

Fig. 12. Example of translating control nodes preceded by a variable into CTL-formulas

5. We translate the control nodes of the model if they are yet not translated starting at the second level (the first level always includes Actions or Guards which are translated into atomic propositions). The variables which are yet not assigned a CTL-formula and followed directly by control nodes, which are in turn directly followed by Actions, Guards or variables already assigned a CTL-formula in terms of Actions or Guards, are assigned the CTL-formula to which the control node is translated according to the translation table of control nodes preceded by variables [21]. Fig. 11 does not encounter this case.

   If we reach a control node, which all incoming and outgoing temporal relationships are attached to Actions, Guards, or variables with assigned values, we translate it according to the translation tables of control nodes dedicated for simple models [21]. For example, we translate the DecisionNode on Level 3 in Fig. 11 depending on the first EPPSL Pattern in Fig. 9:
   AG*(receive application* $\rightarrow$ *(****AF****(X)* $\vee$ ***AF****(make an interview)))*
   We substitute *X* by its value:
   AG*(receive application* $\rightarrow$ *(****AF****((****AF****(refuse the application)* $\wedge$ ***AF****(remove the application data from the system)))* $\vee$ ***AF****(make an interview)))*
   The previous CTL-formula represents the whole complex model in Fig. 11.

Strategy 2: Given an EPPSL complex model, e.g. the complex model of QC_8 in Fig. 13, which we want to verify on the business process model in Fig. 1:

QC_8: *"If there exists a possibility to make an interview, then there exists no possibility to accept the application online and there exists a possibility to make an interview per phone or there exists a possibility to make an interview per Internet".*

We translate the model in Fig. 13 by applying the following steps:

1. Each control node may have either multiple incoming connectors or multiple outgoing connectors. In both cases, we combine the control node with its multiple connectors and their attached ConstraintContainers in one ConstraintContainer. For example, in Fig. 13, we apply this rule on the DecisionNode and the ForkNode resulting in two new ConstraintContainers as shown in Fig. 14.
2. We start to translate the partial quality constraint models which do not include ConstraintContainers. For example, in Fig. 14, we translate (1), (2), (3), and (4) depending on the EPPSL patterns in Fig. 9:
   (1): *Start* $\rightarrow$ ***EF****(make an interview)*
   (2): *Start* $\rightarrow$ $\neg$ ***EF****(accept the application online)*

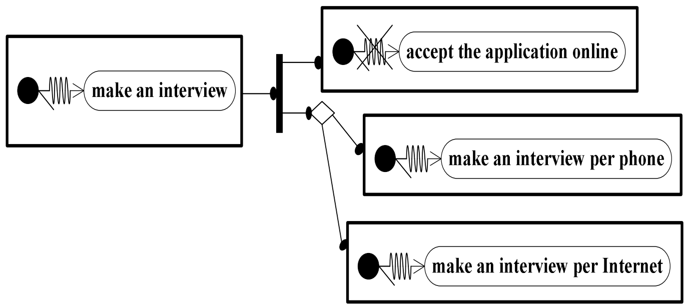

make an interview
accept the application online
make an interview per phone
make an interview per Internet

Fig. 13. Example of a complex model

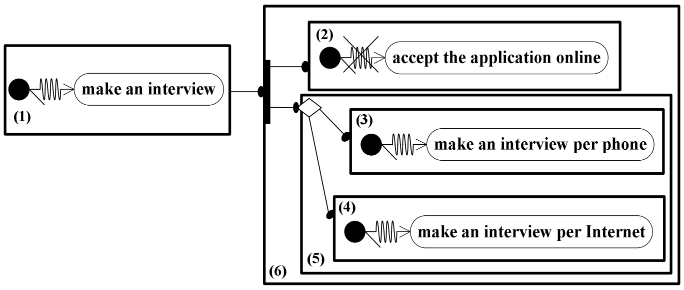

(1)
make an interview
(2)
accept the application online
(3)
make an interview per phone
(4)
make an interview per Internet
(5)
(6)

Fig. 14. Translation strategy of complex models, where control nodes link ConstraintContainers

(3): *Start* $\rightarrow$ ***EF****(make an interview per phone)*
(4): *Start* $\rightarrow$ ***EF****(make an interview per Internet)*

3. We translate the partial quality constraint models which contain ConstraintContainers, which in turn include partial quality constraint models already translated into CTL-formulas. For example, since (3) and (4) in Fig. 14 are already translated, we translate (5) depending on the EPPSL patterns in Fig. 9:
   (5): *(Start* $\rightarrow$ ***EF****(make an interview per phone))* $\vee$ *(Start* $\rightarrow$ ***EF****(make an interview per Internet))*
   Then we translate (6) since (2) and (5) are already translated, depending on the EPPSL patterns in Fig. 9:
   (6): *(Start* $\rightarrow \neg$ ***EF****(accept the application online))* $\wedge$ *((Start* $\rightarrow$ ***EF****(make an interview per phone))* $\vee$ *(Start* $\rightarrow$ ***EF****(make an interview per Internet)))*
4. We always reach a state, where we have two ConstraintContainers, including two translated partial quality constraint models, and a connector, which connects the first to the second one. We translate this state depending on the EPPSL patterns for simple models [21] by stating that the CTL-formula representing the first ConstraintContainer implies the CTL-formula representing the second one. In Fig. 14, we reach a state where (1) implies (6). We translate it depending on the EPPSL patterns in Fig. 9:

(Start $\rightarrow$ ***EF****(make an interview))* $\rightarrow$ *((Start* $\rightarrow \neg$ ***EF****(accept the application online))* $\wedge$ *((Start* $\rightarrow$ ***EF****(make an interview per phone))* $\vee$ *(Start* $\rightarrow$ ***EF****(make an interview per Internet))))*

The previous CTL-formula represents the whole complex model in Fig. 13.

6 Conclusion and Outlook

In this paper, we have introduced a new approach for modeling and formalizing business process quality constraints aiming at providing the users with more flexibility by allowing them to construct constraints with either deterministic or non-deterministic future and by that to enhance the expressiveness ability. Our approach introduces the Extended Process Pattern Specification Language (EPPSL), which is a heavy weight extension of UML Activity Diagrams, and could be easily transformed to be based on any other business process modeling language. EPPSL provides a set of intuitively understandable modeling elements to model quality constraints in terms of branching temporal logic. We also provide a pattern-based translation for EPPSL models into CTL-formulas to achieve the formalization of quality constraints. In our approach, the basic blocks of quality constraints are based only on actions and guards, since these blocks could be actions, guards, anonymous steps (refer to unknown actions or unknown guards), and partial quality constraints (based on the previous three blocks). Later, other basic blocks could be considered, e.g. data objects. In our approach, we only consider future temporal relationships. Past temporal relationships and real-time relationships are open for future work. In our approach, we do not consider the identification of conflicting constraints, which may contain contradicting semantics. Later, this aspect could be considered.

References

1. ISO 9001:2000:Quality Management Systems - Requirements. ISO International Organization for Standardization (2000)
2. Förster, A., Engels, G., Schattkowsky, T., Van Der Straeten, R.: Verification of Business Process Quality Constraints Based on Visual Process Patterns. In: The First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE 2007), pp. 197–208. IEEE Computer Society, Shanghai (2007)
3. Object Management Group:UML 2.0 Superstructure. Version 2.0 (2005), `http://www.omg.org/spec/UML/2.0/Superstructure/PDF/` (last visited 2.12.2010)
4. Förster, A., Engels, G., Schattkowsky, T.: Activity Diagram Patterns for Modeling Quality Constraints in Business Processes. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 2–16. Springer, Heidelberg (2005)
5. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), pp. 46–57 (1977)
6. Emerson, E.A.: Temporal and Modal Logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 955–1072. MIT Press, Cambridge (1990)
7. Liu, Y., Müller, S., Xu, K.: A Static Compliance-Checking Framework for Business Process Models. IBM Systems Journal 46, 335–361 (2007)

8. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Tennenholtz, M. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006)
9. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), Annapolis, Maryland, USA, pp. 287–300 (October 2007)
10. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. Dissertation. TU Eindhoven (2008)
11. Awad, A.:BPMN-Q: A Language to Query Business Processes. In: EMISA 2007. LNI, vol. P-119, pp.115-128. GI (2007)
12. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer, Heidelberg (2008)
13. Laroussinie, F., Schnoebelen, P.: A Hierarchy of Temporal Logics with Past. Theoretical Computer Science 148, 303–324 (1995)
14. Zuck, L.: Past Temporal Logic. PhD thesis. Weizmann Intitute, Rehovet, Israel (1986)
15. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50, 1281–1294 (2008)
16. Wörzberger, R., Kurpick, T., Heer, T.: Checking Correctness and Compliance of Integrated Process Models. In: Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2008), pp. 576–583. IEEE Computer Society, Los Alamitos (2008)
17. Object Management Group: Object Constraint Language (OCL) Specification - Version 2.0 (May 2006), `http://www.omg.org/cgi-bin/doc?formal/2006-05-01` (last visited 2.12.2010)
18. Ly, L.T., Rinderle-Ma, S., Dadam, P.: Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 9–23. Springer, Heidelberg (2010)
19. Hodges, W.: Classical Logic I: First Order Logic. In: Goble, L. (ed.) The Blackwell Guide to Philosophical Logic. Blackwell, Malden (2001)
20. Ghose, A.K., Koliadis, G.: Auditing Business Process Compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
21. Khaluf, L.: Business Process Quality Assurance. Master thesis. University of Paderborn, Paderborn, Germany (May 2010)