# Defining Adaptation Constraints for Business Process Variants

Ruopeng Lu[1], Shazia Sadiq[2], Guido Governatori[3], Xiaoping Yang[2]

[1] SAP Research CEC Brisbane, Australia
ruopeng.lu@sap.com
[2] School of Information Technology and Electronic Engineering
The University of Queensland, Brisbane, Australia
{shazia,xiaoping}@itee.uq.edu.au
[3] NICTA, Australia
guido.governatori@nicta.com.au

**Abstract.** In current dynamic business environment, it has been argued that certain characteristics of ad-hocism in business processes are desirable. Such business processes typically have a very large number of instances, where design decisions for each process instance may be made at runtime. In these cases, predictability and repetitiveness cannot be counted upon, as the complete process knowledge used to define the process model only becomes available at the time after a specific process instance has been instantiated. The basic premise is that for a class of business processes it is possible to specify a small number of essential constraints at design time, but allow for a large number of execution possibilities at runtime. The objective of this paper is to conceptualise a set of constraints for process adaptation at instance level. Based on a comprehensive modelling framework, business requirements can be transformed to a set of minimal constraints, and the support for specification of process constraints and techniques to ensure constraint quality are developed.

**Key words:** Business Process Management, Process Variant Management, Constraint-Based BPM, Business Process Constraint Network

## 1 Background and Motivation

In order to provide a balance between the opposing forces of control and flexibility, we have argued for [12], a modelling framework that allows part of the model that requires less or no flexibility for execution to be predefined, and part to contain loosely coupled process activities that warrant a high level of customization. When an instance of such a process is created, the process model is *concretised* by the domain expert at runtime. The loosely-coupled activities are given an execution plan according to instance-specific conditions, possibly some invariant process constraints, and their expertise.

The foremost factor in designing business processes is achieving improvements in the business outcomes [4]. However, decisions at the strategic level need to be evaluated in light of constraints that arise from several sources. It has been identified that at least four sources of constraints have impact on a business process design:

– **Strategic constraints** define the tactical elements of the process e.g. approval of director required for invoices beyond a certain value.
– **Operational constraints** are determined through physical limitations of business operations, e.g. minimum time for warehouse offloading.
– **Regulatory constraints** are prescribed by external bodies and warrant compliance e.g. financial and accounting practices (Sarbanes-Oxley Act), or batch identification for FDA in pharmaceutical industry.
– **Contractual constraints** define the contractual obligations of an organization towards its business partners, e.g. maximum response time for a service.

In order to harness the full power of BPM techniques, each of these constraints should eventually be translated into constructs of a (executable) business process model, and subsequently lead to process enforcement at the business activity level. This paper will introduce a *design time* modelling approach for *process constraints*. This approach transfers part of the process modelling effort to domain experts who make execution decisions at runtime. Instance adaptation is supported by techniques for specifying instance-specific process models and constraint checking in different variants of the business process [12]. We will demonstrate how the specification of so-called selection constraints can lead to increased flexibility in process execution, while maintaining a desired level of control.

The rest of the paper is organized as follows. We use a running example to motivate the overall approach in section 2. Section 3 provides background concepts for the underlying framework for supporting instance adaptation. In section 4, we introduce the core concept of process constraints. The approach is evaluated against an application scenario in section 5, where the constraint editor prototype is presented. Related work is presented in section 6, followed by the conclusion and future work in section 7.

## 2 Motivating Example

Consider a customer request processing workflow in a CRM (Customer Relationship Management) system. The process is triggered when a customer submits a purchase request. Upon receiving which, a senior sales representative creates a customer requirement document in the system, then he specifies a list of activities to manage the customer request for different workflow roles/participants. The purchase request is finally confirmed and further procurement activities are carried out. The procedures for creating and confirming customer requirement document are predictable and repetitive, while the customer request management activities will typically be case-specific and can be uniquely configured for each customer request which require ad-hoc adjustment. Fig. 1 shows such a workflow in BPMN notation, where the dynamic part of the workflow is modelled as an Ad-Hoc Sub-Process [9].

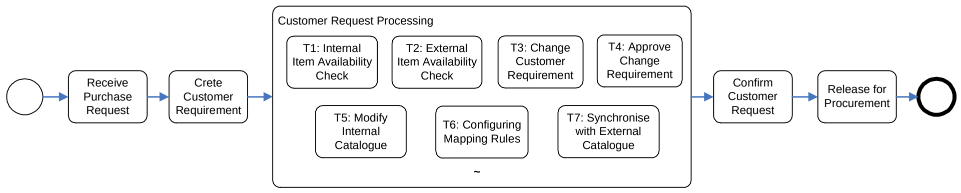

Receive Purchase Request
Crete Customer Requirement
Customer Request Processing
T1: Internal Item Availability Check
T2: External Item Availability Check
T3: Change Customer Requirement
T4: Approve Change Requirement
T5: Modify Internal Catalogue
T6: Configuring Mapping Rules
T7: Synchronise with External Catalogue
~
Confirm Customer Request
Release for Procurement

**Fig. 1.** A network diagnostics scenario modelled in BPMN notation

At the same time, some dependencies and restrictions among the available tests can be abstracted beforehand as follows:

– Internal Item Availability Check $T1$ must be performed for all customer requirement cases;
– Based on operational guideline, maximal 6 activities can be selected for prompt customer response;
– The customer requirement can be adjusted according to case-specific conditions ($T3$), but needs to be approved by a senior sales representative ($T4$);
– The mapping rules can be configured to map content from external catalogue to internal catalogue ($T6$), after which the internal catalogue should be synchronised with the external catalogue to reflect the rule change ($T7$). However, synchronisation $T7$ can be triggered without change in mapping rules;
– In order to avoid inconsistency, manual modification to internal catalogue ($T5$) and automatic synchronisation with external catalogue ($T7$) should not be selected at the same workflow instance.

It is obvious that even with a small number of activities, there is a large number of possible scenarios that satisfy the selection restrictions. For example, the sales representative can decide to execute one, two or up to six activities to fulfil the request processing goals, where the combination of selected activities can be varied. Consider that for each such scenario, the sales representative can further define a variety of execution plans including any execution order and patterns, e.g., execute all selected tasks in sequence, in parallel, or some in sequence and others in parallel (Fig. 2). In graphical modelling notation (e.g., BPMN), it is tedious if not possible to capture and manage all such scenarios and to articulate the conditions for selecting the alternative branches.

## 3 Fundamentals

Constraint Satisfaction are supported by formal methods and its suitability and applicability to solve applied problems such as job-shop scheduling have been extensively demonstrated [2]. The declarative nature of constraint specification,

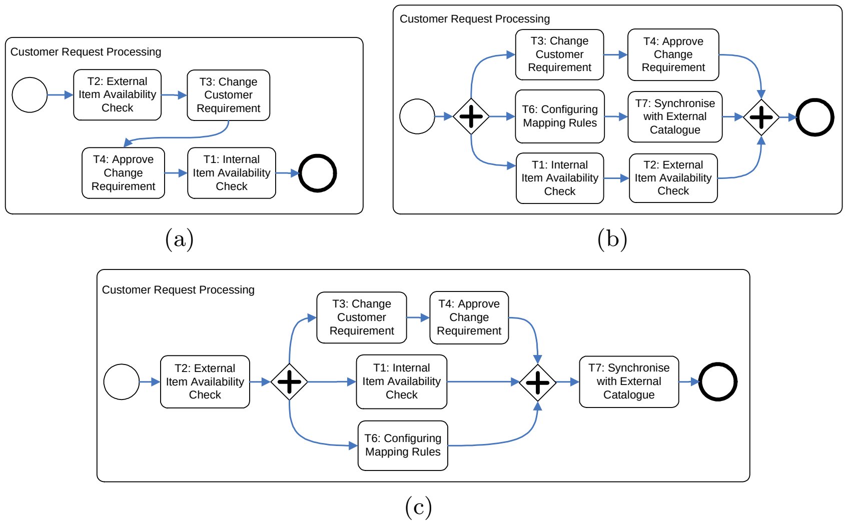

Customer Request Processing
T2: External Item Availability Check
T3: Change Customer Requirement
T4: Approve Change Requirement
T1: Internal Item Availability Check
(a)
Customer Request Processing
T3: Change Customer Requirement
T4: Approve Change Requirement
T6: Configuring Mapping Rules
T7: Synchronise with External Catalogue
T1: Internal Item Availability Check
T2: External Item Availability Check
(b)
Customer Request Processing
T2: External Item Availability Check
T3: Change Customer Requirement
T4: Approve Change Requirement
T1: Internal Item Availability Check
T6: Configuring Mapping Rules
T7: Synchronise with External Catalogue
(c)

**Fig. 2.** Execution possibilities of the BPMN ad hoc sub-process shown in Fig. 1

and the automated reasoning backed by formal theory provides a potential hand-in-gloves solution for the flexible workflow problem defined in this paper.

Nevertheless, applying Constraint Satisfaction in flexible workflows is not straightforward. The overall challenge is to find the right balance between the expressiveness of the constraint language that is rudimentary for expressing constraints in workflows, and the computational efficiency of the resulting constraint networks. The fundamental requirements are to design a constraint language that is expressive enough, while computationally efficient algorithms can be designed and applied for reasoning, and is usable for human constraint designer.

Business process semantics is complex in a sense that it involves different level of abstractions. Control flow, data dependency and resource allocation are the primary aspects that needs to be modelled, among which control flow is the fundamental aspect in workflow models [5]. As such, process constraints should express foremost and at least the restrictions on the structural dependency between workflow activities in a workflow model.

Through intensive investigations, a variant of Constraint Satisfaction have been identified as the theoretical foundation for process constraints, namely, Boolean Constraint Network. Boolean constraints are used to specify the inter-task dependencies, e.g., the effect on other tasks when selecting task A to execute, and should task B be selected if A is depending on B (for logical or data dependency). In particular, the consideration of Boolean constraints is confined within the subset of *tractable* constraints. Tractable constraints are the the class of constraints where polynomial time solution algorithms are known [3]. A mapping has been established between process requirements as directed by business process semantics, and the formal expressions in a subset of Boolean constraints. Then the expressiveness of the constraint language can be determined, which justifies the appropriateness of the constraint language for expressing process requirements. Once the constraint language is defined, appropriate algorithms can be applied for constraint validation, and process design verification.

**Definition 1 (Constraint)** *A constraint $C$ is a pair $(L, R)$, where $L \subseteq X$ is an $m$-tuple of variables, called the scope of the constraint; $X$ is a finite set of variables. $R$ is an $m$-ary relation defined on $L$, which is a subset of the set of all $m$-tuples of elements from some set $D$, called the domain.*

The set of all $m$-tuples $(d_1, d_2, \ldots, d_k)$ of elements from some set $D$ with $d_1 \in D_1, d_2 \in D_2, \ldots, d_k \in D_k$ is the *Cartesian* product $D_1 \times D_2 \times \ldots \times D_k$. The length of the tuple $m$ is called the arity of the constraint. When there is no ambiguity on the scope of the constraints, a relation and a constraint is referred to interchangeably.

For the subsequent discussion on process constraints, we also define this specific term: A process template $PT$ is defined by a set of process tasks $T$ and a set of process constraints $C$. $T$ represent the available pool of activities to be adapted at runtime. $C$ is a set of selection constraints that defines relations between the properties of tasks in $T$.

## 4 Selection Constraints

The adaptation of a process instance is governed by selection and subsequent scheduling constraints. Selecting constraints regulates how and what tasks can be chosen to perform, while scheduling constraints address how these selected tasks are executed, e.g., order of execution, in sequence or parallel [7]. This paper focuses on the first perspective.

### 4.1 Conceptualisation

The following classes of selection constraints have been identified:

Mandatory constraint *man* defines a set of tasks that *must be executed* in every process variant, in order to guarantee that intended process goals will be met.

Prohibitive constraint *pro* defines a set of tasks that *should not be executed* in any process variant.

Cardinality constraint specifies the minimal *minselect* and maximal *maxselect* cardinality for selection among the set of available tasks.

Inclusion constraint *inc* expresses the *dependency* between two tasks $T_x$ and $T_y$, such that the presence of $T_x$ imposes restriction that $T_y$ must also be included. Prerequisite constraint *pre* is the inverse of an inclusion constraint.

Exclusion constraint *exc* prohibits $T_y$ from being included in the process variant when the $T_x$ is selected.

Substitution constraint *sub* defines that if $T_x$ is not selected, then $T_y$ must be selected to *compensate the absence* of the former.

Corequisite constraint *cor* expresses a stronger restriction in that either both $T_x$ and $T_y$ are selected, or none of them can be selected, i.e., it is not possible to select one task without the other.

Exclusive-Choice constraint *xco* is also a more restrictive constraint on the selection of alternative tasks, which requires at most one task to be selected from a pair of tasks $(T_x, T_y)$.

## 4.2 Formalisation

Let $T = \{T_1, T_2, \ldots, T_n\}$ denote the set of all tasks in a process template $PT$. Each task $T_i$ is considered as a propositional variable ranging over domain $D_i = \{0, 1\}$. Let $T_i = 1$ stand for the *presence* of task $T_i$ in a process variant $V$ and $T_i = 0$ stand for *absence*.

Mandatory, prohibitive and cardinality constraints can be defined by restricting the domains of respective tasks. A mandatory task $T_i \in T$ is denoted by $man(T_i)$, where *manselect* is a property of $T_i$ restricting its domain $D_i = \{1\}$. The set of all mandatory tasks in a process template $PT$ is given by:

$$R^{man} = \{T_i \mid man(T_i)\}$$

A task $T_i$ in a process template $PT$ is prohibitive if it is forbidden to be selected in any process variants $V$ of $PT$. A single prohibitive task $T_x$ can be denoted by $pro(T_x)$, where *pro* is a property of $T_x$ restricting $D_x = \{0\}$. The set of all prohibited tasks in a process template $PT$ is given by:

$$R^{pro} = \{T_i| \; pro(T_i)\}$$

A *minselect* constraint is denoted by $R^{min(m)} \subseteq T$, such that $|R^{min(m)}| \geq m$, and $\forall T_i \in R^{min(m)}, D_i = \{1\}$. The *minselect* constraint restricts that every process variant $V$ should contain all tasks in $R^{man}$, and zero, one or more tasks from $(T - R^{man})$. A *maxselect* constraint is denoted by $R^{max(m)} \subseteq T$, such that $|R^{max(m)}| \leq m$, and $\forall T_i \in R^{max(m)}, D_i = \{1\}$.

The mandatory, prohibitive and cardinality constraints are defined by restricting the domain of a single task. On the other hand, inclusion, exclusion, substitution, prerequisite, corequisite and exclusive-choice constraints are binary relations that are defined by restricting the domains of the pair of tasks. According to the nature of the constraints, we call them **containment constraints**. For example, An inclusion constraint $R^{inc}$ is a binary relation on a pair of variables (tasks) $T_x, T_y \in T$, if and only if (*iff*):

$$R^{inc} = ((T_x, T_y), \{(0, 0), (0, 1), (1, 1)\})$$

An inclusion constraint $R^{inc}$ defined on tasks $T_x, T_y$ reads $T_x$ includes $T_y$. By definition, it restricts the domain of values that can be assigned to the pair $(T_x, T_y)$. In this case, either $(0, 0)$, $(0, 1)$, or $(1, 1)$ can be assigned. Applying this definition to task selection, it expresses that when $T_x$ is selected, $T_y$ must also be selected ($T_x$ is the dependent of $T_y$). The following selection scenarios are permitted:

– neither $T_x$ nor $T_y$ is selected, i.e., (0, 0);
– $T_y$ is selected without $T_x$, i.e., (0, 1);
– both $T_x$ and $T_y$ are selected, i.e., (1, 1).

The scenario $(1, 0)$ is prohibited where $T_x$ is selected without $T_y$, thus enforcing the inclusion relationship between selection of $T_x$ and $T_y$.

Similarly, an exclusion constraint $R^{exc}$ is a binary relation on a pair of variables $T_x, T_y \in T$, *iff*:

$$R^{exc} = ((T_x, T_y), \{(0, 0), (0, 1), (1, 0)\})$$

An exclusion constraint prohibits the selection scenario $(1, 1)$ where both $T_x$ and $T_y$ are selected. Table 1 presents a summary for the definition of the containment constraints.

**Table 1.** Definitions of Containment Constraints

| Constraint | Definition |
|---|---|
| $R^{inc}$ | $((T_x, T_y), \{(0, 0), (0, 1), (1, 1)\})$ |
| $R^{exc}$ | $((T_x, T_y), \{(0, 0), (0, 1), (1, 0)\})$ |
| $R^{sub}$ | $((T_x, T_y), \{(0, 1), (1, 0), (1, 1)\})$ |
| $R^{pre}$ | $((T_x, T_y), \{(0, 0), (1, 0), (1, 1)\})$ |
| $R^{cor}$ | $((T_x, T_y), \{(0, 0), (1, 1)\})$ |
| $R^{xco}$ | $((T_x, T_y), \{(0, 1), (1, 0)\})$ |

For example, the restrictions for customer request processing discussed in section 2 can be expressed as follows:

– *Mandatory* task $T1$, i.e., $man(T1)$
– *Maximal* selection of 6 tasks, i.e., $R^{max(6)}$
– $T3$ and $T4$ are *co-requisite*, i.e., $T3$ *cor* $T4$
– $T6$ *includes* $T7$, i.e., $T6$ *inc* $T7$
– $T5$ and $T7$ are *exclusion*, i.e., $T5$ *exc* $T7$

Considering constraint $T3$ *cor* $T4$, where it regulates that either both $T3$ and $T4$ are selected in the same workflow instance, or none of them should be. Similarly, the substitution constraint $T5$ *sub* $T7$ prohibits the case when both $T5$ and $T7$ are selected at the same instance.

## 4.3 Expressiveness of Selection Constraints

A constraint language $L$ is defined by imposing restrictions on the possible constraint relations allowed to use. The expressiveness of the selection constraint language can be investigated through the inverse, intersection and composition of the abovementioned selection constraints. These constraint properties will also be utilised in the validation algorithm later in the paper.

**Inverse** Given a containment constraint $R$, the ***inverse*** $R^{-1}$ of $R$ also follows basic set theoretic definition, i.e.,

$$R^{-1} = \{(a, b) | (b, a) \in R\} \tag{1}$$

The inverse of a prerequisite constraint is equivalent to an inclusion constraint. The rest of the four containment constraints are reflexive, therefore their inverse relations equal to themselves. The inverse of a mandatory constraint is the prohibitive constraint and vice versa.

**Intersection** For (binary) containment constraints, it is allowed for more than one selection constraint to be specified on the same pair of tasks (variables). The ***intersection*** between two containment constraints $R_1$ and $R_2$ defined on the same pair of variables $(T_x, T_y)$ is denoted by $R_1 \cap R_2$. Intersections between selection constraints follow the classic set theoretic definition, i.e.,

$$R_1 \cap R_2 = \{(a, b) | (a, b) \in R_1,\ (a, b) \in R_2\} \tag{2}$$

For example, suppose an inclusion and a substitution constraint are defined on tasks $T_x$ and $T_y$, i.e., $T_x$ *inc* $T_y$, and $T_x$ *sub* $T_y$. The set of possible values for $(T_x, T_y)$ that satisfies both *inc* and *sub* is given by:

$$inc \cap sub = \{(0, 0), (0, 1), (1, 1)\} \cap \{(0, 1), (1, 0), (1, 1)\} = \{(0, 1), (1, 1)\}$$

The resulting relation requires $T_y$ to be a mandatory task since $T_y = 1$ in both cases. Furthermore, the intersection between containment constraints and the mandatory and prohibitive constraints can also be defined.

**Composition** The ***composition*** of two containment constraints $R(T_x, T_y)$ and $R(T_y, T_z)$, denoted by:

$$R(T_x, T_y) \otimes R(T_y, T_z)$$

results in the binary relation $R(T_x, T_z)$, where

$$\begin{aligned} R(T_x, T_z) &= R(T_x, T_y) \otimes R(T_y, T_z) \\ &= \{(a, b) | a \in D_x, b \in D_z, \exists c \in D_y, (a, c) \in R(T_x, T_y), (c, b) \in R(T_y, T_z)\} \end{aligned} \tag{3}$$

In order to make mandatory and prohibitive relations union-comparable to containment relations, the relation $R(T_x, T_y)$ between an unconstrained task, say $T_x$, and a mandatory task $T_y$ can be denoted as $\{(0, 1), (1, 1)\}$, which allows any value of $T_x$ as long as $T_y = 1$. Similarly, if $T_y$ is prohibitive, and $T_x$ is unconstrained, $R(T_x, T_y) = \{(0, 0), (1, 0)\}$. For example, suppose defining $T_x$ *pre* $T_y$ on an arbitrary pair of tasks $(T_x, T_y)$, while $T_y$ is mandatory. It implies that both $T_x$ and $T_y$ to be mandatory, i.e.,

$$R(T_x, T_y) = \{(0, 0), (1, 0), (1, 1)\} \cap \{(0, 1), (1, 1)\} = \{(1, 1)\}$$

While if $T_x$ is prohibitive, and $T_x$ *pre* $T_y$ for tasks $T_x$ and $T_y$, it can be inferred that $T_y$ is also prohibitive.

## 4.4 Validation of Selection Constraints

Given the definition of selection constraints in a process template $PT$, it is necessary to make sure at design time that:

– all implicit constraints are made explicit; and
– no conflict exists in the constraint specification.

In order to formally address the problem, a Selection Constraint Network (SCN) is defined to provide formal semantics to validate the selection constraints. SCN is a binary Boolean constraint network [6], which is a special type of constraint network with variables having bi-valued domains 1, 0 or true, false. Validation of selection constraints of a process template $PT$ is realized by checking for the consistency of the corresponding SCN.

**Definition 2 (SCN)** *Given a process template $PT$, $SCN$ is a binary constraint network defined by $(X^s, D^s, C^s)$. $X^s = \{T_1, \dots, T_n\}$ is the set of all tasks in PT represented as propositional variables. $D^s = \{D_1, \dots, D_n\}$ is the set of domains of the corresponding propositional variables, where $D_i \in D^s, D_i \subseteq \{0, 1\}$ . $C^s$ is the set of all selection constraints in $PT$.*

For example, a $SCN = (X^s, D^s, C^s)$ can be formulate for the the selection constraints modelled for the customer request processing workflow in section 4, where:

– $X^s = \{T1, \dots, T7\}$
– $D^s = \{D1, \dots, D7\}$, where

$$Di = \begin{cases} \{1\} & i \in \{1\} \\ \{0,1\} & i \in \{2,3,4,5,6,7\} \end{cases}$$

– $C^s = \{R(T3, T4), R(T5, T7), R(T6, T7)\}$, where
  – $R(T3, T4) = T3\ cor\ T4$,
  – $R(T5, T7) = T5\ inc\ T7$, and
  – $R(T6, T7) = T6\ exc\ T7$

A *solution* to a SCN is the assignment of values to each variable such that no constraint is violated, i.e., $\forall T_i \in X^s$, there is a consistent assignment of a value from $D_i$ such that all relevant selection constraints in $C^s$ are satisfied. A solution corresponds to a task selection scenario for the process template $PT$ that satisfies all relevant selection constraints in $PT$. A SCN is said to be *consistent* if there exists at least one solution. If conflict exists between the constraints, the constraint network is inconsistent and hence no solution exists.

Given an SCN as a binary Boolean constraint network, the consistency of SCN can be checked by applying a generic consistency checking algorithm to SCN (cf. Fig. 3 adapted from [3]).

The algorithm checks whether each pair of constraints in SCN is consistent. If conflicts exist, the operation $R(T_i, T_j) \cap (R(T_i, T_k) \otimes R(T_k, T_j))$ results in an

**Fig. 3.** Consistency checking procedure for SCN

```
Procedure SCN Consistency
Input: A constraint network SCN $(X^s, D^s, C^s)$
Output: A consistent constraint network $SCN'$ equivalent to $SCN$
Method:
  do
    $Q \leftarrow SCN$
    for each $k$ from 1 to $n$ do
      for each $i, j$ from 1 to $n$, $R(T_i, T_j) \in Q$ do
        $R(T_i, T_j) \leftarrow R(T_i, T_j) \cap (R(T_i, T_k) \otimes R(T_k, T_j))$
        if $R(T_i, T_j) = \emptyset$ then break
  until $Q = SCN$
```

empty constraint $R(T_i, T_j)$ for some $T_i$ and $T_j$, and the checking algorithm terminates. Conflicts can exist between containment constraints, or between mandatory and containment constraints. However, if no conflict is detected, further composition operation will not change the constraints in SCN, and the algorithm terminates. Furthermore, implicit constraints will be inferred and added to SCN, as the result of applying the composition operation to each pair of tasks.

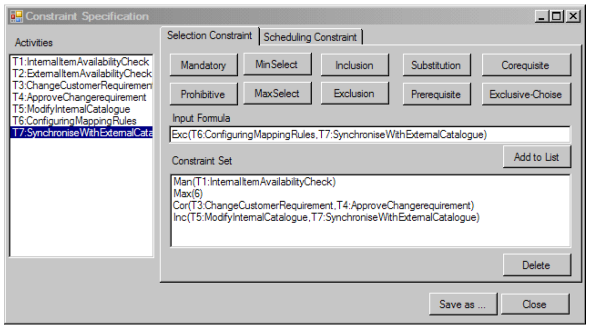
Considering the following example to help explain how the algorithm is applied to reason about the selection constraints. Applying the algorithm to the given $SCN$ example, a more restrictive relation between $(T5, T6)$ is obtained by applying composition $\otimes$ to $R(T5, T7)$ and $R(T6, T7)$. We take the inverse of $R(T6, T7)$, where $R^{-1}(T6, T7) = R(T7, T6) = T7\ pre\ T6$. Applying composition operation $R(T5, T7) \otimes R(T7, T6)$, the new relation $R(T5, T6) = T5\ exc\ T6$ is obtained (cf. equations (1) and (3) in section 4.3). Since no conflict is detected, the example $SCN$ is consistent[1]. Hence, the specification of selection constraints for the customer request processing workflow is correct.

## 5 Prototypical Implementation

We have implemented a prototype to evaluate the theoretical constraint modelling approach. An extensible constraint designer has been built for end users to model selection constraints (and other constraint types such as scheduling constraints [7]). Fig. 4 is a snapshot of the prototype showing the modelled constraints from the previous examples. A built-in function is provided for design time constraint validation. If validation is successful, the editor exports the constraint specification into a XML file, which can be deployed into a dynamic workflow engine called Chameleon. Chameleon is a light-weight workflow platform built upon Windows Workflow Foundation that supports flexible workflow execution such as Pocket of Flexibility [12]. The deployed constraints file is used

[1] Note that the cardinality constraint can be simply checked by counting the number of selected tasks in $X^s$.

to automatically verify instance adaptation for different process instances on the Chameleon platform during instance adaptation.



T1:InternalItemAvailabilityCheck
T2:ExternalItemAvailabilityCheck
T3:ChangeCustomerRequiremen
T4:ApproveChangerequirement
T5:ModifyInternalCatalogue
T6:ConfiguringMappingRules
T7:SynchroniseWithExternalCata

Exc(T6:ConfiguringMappingRules,T7:SynchroniseWithExternalCatalogue)

Man(T1:InternalItemAvailabilityCheck)
Max(6)
Cor(T3:ChangeCustomerRequirement,T4:ApproveChangerequirement)
Inc(T5:ModifyInternalCatalogue,T7:SynchroniseWithExternalCatalogue)

**Fig. 4.** Constraint Editor Prototype

## 6 Related Work

The proposed constraint-based approach falls in the category of *late binding* paradigm, which is an approach where parts of the process schema are left undefined at design time, but are configured at runtime for each process instance. For this purpose, placeholder activities are provided, which are modelled and executed during run-time. The guidelines or restrictions on the runtime adaptation are modelled as rules [1, 8] or constraints [12]. On the other hand, there have been a number of constraint languages proposed in other disciplines, such as Object Constraint Language (OCL) [10] for object-oriented analysis and design method. OCL is based on first-order predicate logic but it uses syntax similar to programming languages. As a result, it is used to express additional constraints on UML models that cannot specified with the graphical means provided by UML. OCL has been applied to model resource allocation in workflows. However OCL and UML do not support concepts usually adopted in the characterisation of control flows requirement in workflows. At the same time, the DECLARE approach [11] has a similar problem-solving philosophy, which also aims at supporting instance-level process adaptation by defining a set of workflow constraints to regulate flexible changes. The background theory of DECLARE constraints is Linear Temporal Logic (LTL), which operates on additional temporal logical operators including *always*, *eventually*, *until* and *next time*. As a result, the focus of DECLARE constraints has been on the temporal dependencies. In our constraint framework, late binding is seen as a two-step approach. At runtime, domain expert adapts a process instance by first selecting a set of tasks to perform, and secondly structuring the tasks into certain workflow patterns. As a result, two constraints systems have been developed to support such activities. This paper has focused on modelling the dependencies on task selection, as the first and most essential step in late binding paradigm. The structuring (scheduling) constraints have been covered by our previous work [7].

## 7 Conclusion and Outlook

Process constraints can express minimal restrictions on the selection and ordering of tasks for all instances of the targeted business process, thus providing a degree of flexibility in process execution. This paper has presented how task selection constraints can be specified at *design time*, through selection constraints, and the quality of the constraint specification is checked through the formal machinery of selection constraint network respectively. Different process models can be built/tailored for individual process instances at runtime, leading to instance-specific process models (process variants). Possible future work includes to explore further constraint dimensions such as the resource perspective, and a complete implementation of the extended constraint sets.

## References

1. Adams, M., ter Hofstede, A. H. M., Edmond, D., van der Aalst., W. M. P.: Implementing Dynamic Flexibility in Workflows using Worklets. BPMcenter.org (2006)
2. Bartk, R.: Dynamic constraint models for planning and scheduling problems. In Lecture Notes in Computer Science, Vol. 1865, Springer-Verlag (1999) 237-255
3. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
4. Indulska, M., Chong, S., Bandara, W., Sadiq, S., Rosemann, M.: Major issues in business process management: A vendor perspective. In Proc. the Pacific Asia Conference on Information Systems (PACIS2007) (2007)
5. Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. In Proc. Int'l Working Conference on Dynamic Modelling and Information Systems, Nordwijkerhout 1994 (1994)
6. Jeavons, P.: Constructing constraints. In Proc. Int'l Conference on Principles and Practice of Constraint Programming. Springer-Verlag (1999)
7. Lu, R., Sadiq, S., Padmanabhan, V., Governatori, G.: Using a Temporal Constraint Network for Business Process Execution. In Proc. 17th Australasian Database Conference (ADC2006), Hobart, Australia (2006)
8. Muller, R., Greiner, U., Rahm, E.: AGENT WORK: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng., Elsevier Science Publishers B. V., Vol.51 (2004) 223-256
9. Object Management Group: Business Process Modeling Notation (BPMN) Specification 1.0 Technical Report, Object Management Group (OMG) (2006)
10. Object Management Group: Object Constraint Language Specification Version 1.1. Technical Report, Object Management Group (OMG) (1997)
11. Pesic, M., Schonenberg, M. H., Sidorova, N., van der Aalst, W. M. P.: Constraint based workflow models: Change made easy. In Proc. OTM Confederated International Conferences 2007 (2007) 77-94
12. Sadiq, S., Sadiq, W., Orlowska, M.: A Framework for Constraint Specification and Validation in Flexible Workflows. Information Systems, Vol.30(5) (2005)