# Supervision of Constraint-Based Processes: A Declarative Perspective

Sauro Schaidt, Eduardo de Freitas Rocha Loures,
Agnelo Denis Vieira, and Eduardo Alves Portela Santos

Pontificia Universidade Catolica do Parana, Industrial Engineering,
Imaculada Conceicao 1155, 80215 901 Curitiba, Brazil
{sauro.schaidt,agnelo.vieira,eduardo.loures,
eduardo.portela}@pucpr.br
http://www.pucpr.br/

**Abstract.** Constraint-based processes require a set of rules that limit their behavior to certain boundaries. Declarative languages are better suited to modeling these processes precisely because they facilitate the declaration of little or no business rules.These languages define the activities that must be performed to produce the expected results but not define exactly how these activities should be performed. The present paper proposes a new approach to deal with constraint-based processes. The proposed approach is based on supervisory control theory, a formal foundation for building controllers for discrete-event systems.The controller proposed in this paper monitors and restricts execution sequences of tasks such that constraints are always obeyed. We demonstrate that our approach can be used as a declarative language for constraint-based processes.

**Keywords:** constraint-based processes, declarative languages.

## 1 Introduction

Constraint-based processes require a set of rules that limit their behavior to certain boundaries. Declarative languages are better suited to modeling these processes precisely because they facilitate the formal declaration of such business rules [1] [2] [3]. These languages define the tasks that must be performed to produce the expected results but not define exactly how these activities should be performed. Thus, any execution order of tasks is possible provided that the constraints (imposed by the rules) are not violated. On the other hand, there are processes that require a strong imposition of rules for their implementation. These processes are called rigid or highly structured processes. Imperative languages are better suited for modeling these processes because, unlike declarative languages, they define exactly how a set of tasks should be performed. Thus, we need a model that explicitly defines the ordering and execution of activities.

An imperative model focuses on specifying exactly how to execute the process, i.e., all possibilities have to enter into the model by specifying its control-flow. Procedural (or imperative) models (e.g. BPMN, EPC, YAWL, WF-Nets or

Y.T. Demey and H. Panetto (Eds.): OTM 2013 Workshops, LNCS 8186, pp. 134–143, 2013.

Petri Nets), provide constructs such as AND/OR-splits, AND/OR-Joins, etc. A declarative model specifies a set of constraints, i.e., rules that should be followed during the execution. In this way, the declarative model implicitly defines the control-flow as all possibilities that do not violate any of the given constraints.

Many approaches has proposed to add control to the business processes and to avoid incorrect or undesirable executions of the tasks. A stream of research proposes using rule-based or constraint-based modeling languages [11] [12] [13]. DECLARE is developed as a constraint-based system and uses a declarative language grounded in Linear Temporal Logic (LTL) [4] for the development and execution of process models[2] [3]. With DECLARE, constraints determine activity order, and thus, any order of execution sequence is possible as long as constraints go un-violated.

In the present paper we propose a new approach to deal with constraint-based processes founded on the Supervisory Control Theory [5]. The new approach proposes a control system based on modular supervisors [6] which restrain the process to not violate the constraints. This action is accomplished through dynamic disabling of some events in order to restrain the state space of process. We consider that there may contain substrings that are not allowed. These substrings may violate a desired ordering of events and they need to be avoided. Thus, a modular supervisor is built in order to ensure that the whole set of constraints is not violated. This approach is declarative because it does not limit the user by imposing rigid control-flow structures. In fact, the basis of this approach is to inform users of which tasks are not allowed after an observed trace of events at run-time, and users operate with some freedom because they choose execution sequences allowed under supervision.

According to [7], constraints or business rules include the following aspects: ordering-based (i.e., execution order of tasks in cases), agent-based (i.e., involvement of a role or agent in cases and processes), and value-based (i.e., forms belonging to a task). In the present paper we consider only the ordering-based constraints.

## 2 Supervisory Control

Supervisory control theory (SCT) [5] has been developed in recent decades as an expressive framework for the synthesis of control for Discrete-Event systems (DES). The SCT is based on automata theory, or dually formal language theory, depending on whether one prefers an internal structural or external behavioral description at the start. In SCT, the behavior of a DES is modeled by an automaton. The restrictions to be imposed on the DES can be expressed in terms of a language representing the admissible behavior and it is named specification [5]. The SCT provides computational algorithms for the synthesis of a minimally restrictive supervisor that constrains the behavior of the DES by disabling some events in such a way that it respects the admissible language and that it ensures nonblocking, i.e., there is always an event sequence available to complete a task (or to reach a marked state in terms of automata theory).

Let a Constraint-based Process (CP), modeled at the untimed (or logical) level of abstraction, and whose behavior must be restrained by supervisory control in order to not violate a given set of constraints. Let us assume that the given CP is modeled by automaton $G$, where the state space of $G$ need not be finite. Let $\Sigma$ be the event set of $G$. Automaton $G$ models the uncontrolled behavior of the CP. The premise is that this behavior is not satisfactory and must be modified by control; modifying the behavior is to be understood as restricting the behavior to a subset of $L(G)$. In order to restrain the behavior of $G$ we introduce a supervisor; supervisor will be denoted by $S$. The language $L(G)$ contains strings that are not acceptable because they violate some constraint or nonblocking condition that we wish to impose on the CP. It could be that certain states of $G$ are undesirable and should be avoided. These could be states where $G$ blocks, via deadlock or livelock; or they could be states that are inadmissible. Moreover, it could be that some strings in $L(G)$ contain substrings that are not allowed because they may violate a desired ordering of certain events. Thus, we will be considering sublanguages of $L(G)$ that represent the legal or admissible behavior for the CP.

SCT consider a very general control paradigm for how $S$ interacts with $G$. In this paradigm, $S$ sees (or observes) some, possibly all, of the events that $G$ executes. Then, $S$ tells $G$ which events in the current active event set of $G$ are allowed next. More precisely, $S$ has the capability of disabling some, but not necessarily all, feasible events of $G$. The decision about which events to disable will be allowed to change whenever $S$ observes the execution of a new event by $G$. In this manner, $S$ exerts dynamic feedback control on $G$. The two key considerations here are that $S$ is limited in terms of observing the events executed by $G$ and that $S$ is limited in terms of disabling feasible events of $G$. Thus, it is considered the presence of observable events in $\Sigma$ - those that $S$ can observe - and the controllable events in $\Sigma$ - those that $S$ can disable.

In SCT, the event set of $G$ is partitioned into two disjoint sets, being the set of controllable events $\Sigma_c$, and the set of uncontrollable events $\Sigma_u c$. An event is classified as controllable if its occurrence can be disabled by supervisor. It is classified as uncontrollable in the opposite case. An event might be modeled as uncontrollable because it is inherently unpreventable. Formally, a supervisor $S : L(G) \to 2^{\Sigma}$ is a function that maps from the sequence of generated events to a subset of controllable events to be enabled or disabled. The optimal behavior of the process $G$ under supervision of supervisor $S$ is represented by the language marked by the automaton $S/G$.

## 3 Modeling of Constraint-Based Processes

In order to apply the SCT for controlling constraint-based process, it is necessary to obtain two models: (1) the model of the system under control and (2) the model of constraints. We consider that a constraint-based process is constituted of a set of tasks. In our approach each task is assigned to an automaton representing its behavior. This automaton represents the states through which

a work item passes during execution of such process. This automaton models relevant aspects that will be considered for supervisory control: the beginning, ending and the cancelation of a task. We assume each task $t_i$ (i = 1, 2,,n) is modeled as an automaton with two states: (1) an initial state means the task is not being executed (a case has not entered) and (2) another state means a case is being processed. With event start ($t_{is}$), the task is initiated (state 1 is reached). When it finishes, signaled by the occurrence of event complete ($t_{ic}$) or cancel ($t_{ix}$), it returns to initial state. According to this model, a task may be executed repeatedly over the same instance. We consider that only the beginning of a task is a controllable event. It means the control action is only possible before a task has been initiated. After a task begins, the supervisor can not avoid its ending or canceling. Fig. 1 shows the generic automaton that represents a task $t_i$.

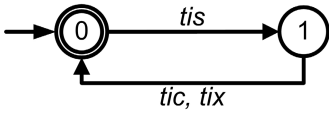
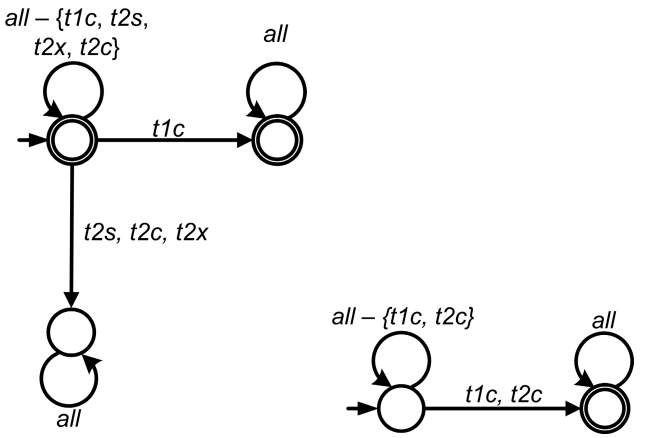



**Fig. 1.** Automaton representing a task $t_i$

The tasks involved in a constraint-based process will be considered the system under control. To represent the behaviour of such system, the synchronous composition between automata is the solution. In this case, we obtain the uncontrolled behaviour of the constraint-based process. To calculate the supervisor that avoids undesired sequences (which violate constraints), it is necessary to express constraints in terms of automata. The set of constraints presented in [9] is considered as a start point for building our set of automata.

In the previous work [8] we propose four groups of constraints and each constraint is modeled by a formal model as in [9]: (1) existence, (2) relation, (3) negation and (4) choice. Existence models specify how many times or when one activity can be executed. Relation models define some relation between two (or more) activities. Negation models define a negative relation between activities. Choice models can be used to specify that one must choose between activities. Considering the application of SCT, we use automata to represent the whole set of constraints. Instead of modeling in LTL as in [9], these four groups are modeled by automata. Fig. 2 shows two examples of constraints over the execution of two tasks $t_1$ and $t_2$ (the reader may check the whole set of constraints in [8]). The precedence model requires that task $t_2$ is preceded by task $t_1$, i.e., it specifies that task $t_2$ can be executed only after task $t_1$ is executed. In this case, other tasks can be executed between tasks $t_1$ and $t_2$. The $1of2$ model specifies that at least one of the two tasks $t_1$ and $t_2$ has to be executed, but both can be executed and each of them can be executed an arbitrary number of times.




(a) Constraint precedence model (b) Constraint 1of2 model

**Fig. 2.** Examples of constraints

## 4 Example of Application

As an example to illustrate our approach, we use the same process treated in [3]. It is a process for handling a patient at the first aid department in a hospital with a suspicion of a fracture. Fig. 3 shows the activities of such process depicted as boxes. There are four constraints to be imposed to this process. Constraint init $C1$ specifies that task examination must be the first executed task in an instance. A specialist may ask for a X-ray to additional diagnosis. Depending on the absence, presence and type of fracture, there are several types of treatment available, such as sling, fixation, surgery and cast. Except for cast and fixation, which are mutually exclusive (constraint not co-existence $C4$), the treatments can be given in any combination and each patient receives at least one treatment (1of4 constraint $C3$). Additional diagnosis (X-ray) is not necessary when the specialist diagnoses the absence of a fracture during examination. Without this additional diagnosis, the patient can only receive the sling treatment. All other treatments require X-ray to rule out the presence of a fracture, or to decide how to treat the fracture (constraint precedence $C2$). Simple fractures can be treated just by cast. Moreover, the specialist can provide medication at any stage of the treatment. Also additional examinations and X-rays can be done during the treatment.

We assume each task $t_i$ (i=1,,7) of the process for handling patient is modeled by an automaton as shown in Fig. 3. Fig. 4 shows the constraints $C1$, $C2$, $C3$ and $C4$ modeled by automata. Considering that we use a modular approach of SCT [6], one supervisor is synthesized in order to satisfy each constraint. The first step to synthesizing modular supervisors is to obtain the local plant for each constraint. Local plants for $C1$, $C2$, $C3$ and $C4$ are given by

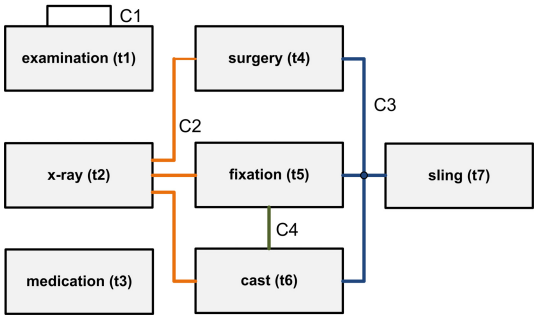
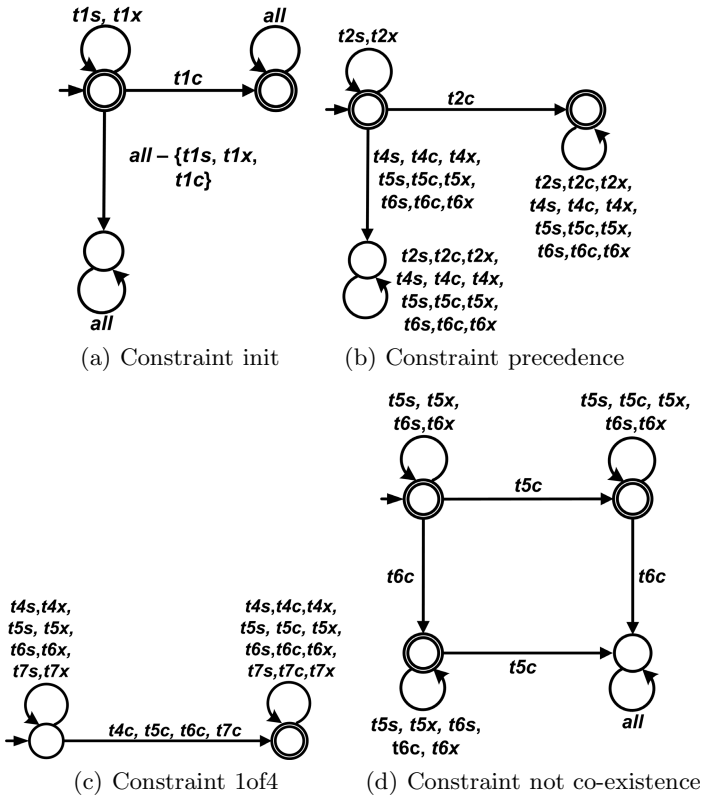



**Fig. 3.** Process for handling a patient at the first aid




(a) Constraint init (b) Constraint precedence

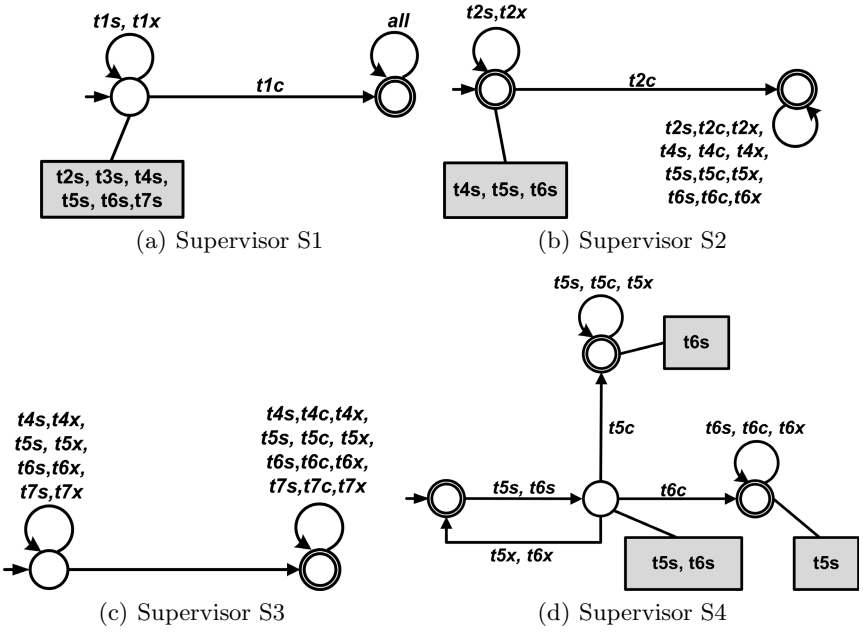
(c) Constraint 1of4 (d) Constraint not co-existence

**Fig. 4.** Examples of constraints

$Gl_{C1} = t1||t2||t3||t4||t5||t6||t7$, $Gl_{C2} = t2||t4||t5||t6$, $Gl_{C3} = t4||t5||t6||t7$, and $Gl_{C4} = t5||t6$, respectively. Synthesis of local supervisor $Sj$ is performed considering corresponding constraint $Ck$ (k=1,2,3,4) and its local plant $Gl_k$. Using algorithms proposed by [5] and [10], the four modular supervisors are obtained, each one guaranteeing that constraints will not be violated.

Each modular supervisor show in Fig. 5 disables a set of controllable events according to its states. A corresponding pair $(Sj, \Phi_j)$ represents each supervisor, where $\Phi_j$ represents the output map. Considering local supervisors Sj shown in Fig. 5, their output maps are: $S_1$ is $\Phi_1(0) = t2s, t3s, t4s, t5s, t6s, t7s$, $\Phi_1(1) = \emptyset$; $S_2$ is $\Phi_2(0) = t4s, t5s, t6s$, $\Phi_2(1) = \emptyset$; $S_3$ is $\Phi_3(0) = \emptyset$, $\Phi_3(1) = \emptyset$; and $S_4$ is $\Phi_4(0) = \emptyset$, $\Phi_4(1) = t5s, t6s$, $\Phi_4(2) = t6s$, $\Phi_4(3) = t5s$. The aim of each modular supervisor is to restrain a set of tasks to a limited state space (defined by each constraint). Thus, the related constraint is not violated. In Fig. 5, the box attached to each state represents the set of disabled events (the output map of each modular supervisor).

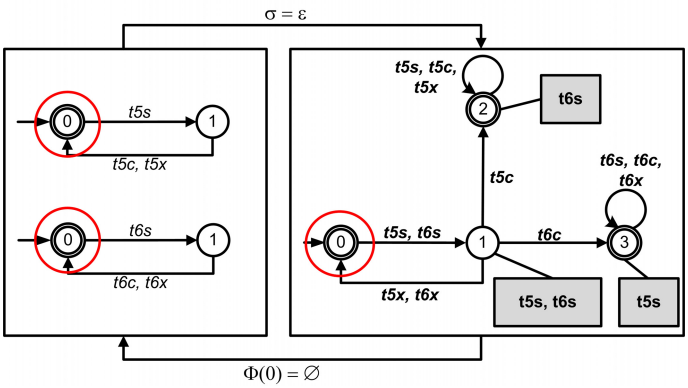
Fig. 6 shows the interplay between tasks $t5$ and $t6$ and the modular supervisor $S4$. In the initial state $S4$, there is no controllable event being disabled ($\Phi_4(0) = \emptyset$). Thus, both tasks $t5$ and $t6$ may be initiated (as shown in Fig. 6(a)). It means users are able to choose one of them to initiate. In the case $t5$ has been chosen, the state 1 of $S4$ is reached and a new output map is established. Fig. 6(b) illustrated this situation. In state 1 of $S4$, the controllable events $t5s$ and $t6s$ are disabled ($\Phi_4(1) = t5s, t6s$). It means users can not initiate task $t6$ unless task



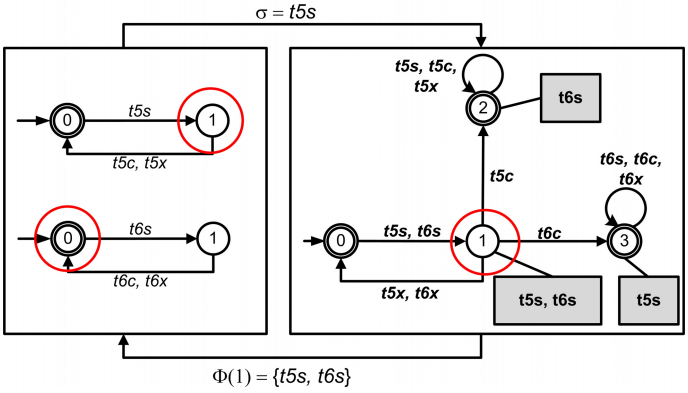

(a) Supervisor S1 (b) Supervisor S2

(c) Supervisor S3 (d) Supervisor S4

**Fig. 5.** Models of supervisors




(a) Interplay in initial state




(b) Initiation of task $t5$

**Fig. 6.** Interplay between tasks $t5$ and $t6$, and supervisor $S4$

$t5$ be canceled. By completing the execution sequence based on output map of supervisor $S4$, users are guaranteed no violation of constraint not co-existence between tasks $t5$ and $t6$.

The control action imposed by modular supervisors allows for many execution paths. Using this approach it is not necessary to include or represent these paths explicitly. For example, the not co-existence constraint between tasks cast ($t5$) and fixation ($t6$) is difficult to express in imperative languages, considering that the choice between these two treatments is not fixed. In an imperative language one would need to decide on the moment of choice, specify the loop behavior, and determine the people making these choices. In the proposed approach, the modular supervisor $S4$ only tells which tasks are allowed to beginning (in some active set of enabled tasks). It does not force which task has to initiate. Instead, the choice is made by users. Also, the behaviour imposed by $1of4$ constraint allows for multiple execution paths. In this case it is necessary a high flexibility of execution of tasks surgery ($t4$), fixation ($t5$), cast ($t6$) and sling ($t7$). The aim

is that a patient may receive any of these treatments in any combination and at least one of them. The modular supervisor $S3$ just tell us that any sequence is possible but a marked state is only reached after the ending of one of the treatments. It means while one of these tasks does not finish, the constraint remains violated. Moreover, after the ending of one of the treatments, any combination of the available treatments is possible.

## 5 Conclusion

Supervisory control theory allows an automatic synthesis of supervisors that the constraints are not violated in a minimally restrictive way and ensures that this behavior is non-blocking (i.e., there is always an event sequence available to complete an activity). Thus, new control actions may be rapidly and automatically designed when modifications, such as redefinition of constraints or tasks arrangements, are necessary. The constraint-based processes can be made to behave optimally with respect to a variety of criteria, where optimal means in minimally restrictive way. Among the criteria are safety specifications like the avoidance of prohibited regions of state space, or the observation of services priorities; and liveness specifications, as least in the weak sense that distinguished target states always remain reachable. Thus, the obtained solution using SCT is correct by construction.

The control approach presented in this paper aims to monitor and restrict execution sequences of tasks such that constraints are not violated. Despite the control logic is built based on constraints, it does not limit the user by imposing rigid control-flow structures. In fact, the basis of our approach is to inform users of which tasks are not allowed after an observed trace of events at run-time, and users operate with some freedom because they choose execution sequences allowed under supervision. Users can adopt this service as a guide to execute tasks with a guarantee that constraints are followed and goals are met. The control approach presented here also offers flexibility to users to choose execution sequences, and it is even possible for users to execute tasks simultaneously with no rules violations.

In constraint-based process is difficult to envision all possible paths and the process are driven by user decisions rather than system decision. Here processes are less repetitive and the emphasis is on flexibility and user empowerment. On the other hand it is difficult to model more abstract relations between tasks when the user has many choices in each state. So, a formal foundation to deal with constraint-based processes is very welcome. Despite the many theoretical results and innovative prototypes, few of the research ideas have been adopted in commercial systems. In fact this is a limitation of the use of the SCT so far.

## References

1. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)

2. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
3. Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development 23(2), 99–113 (2009)
4. Clarke Jr., A., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
5. Ramadge, P., Wonham, W.: The control of discrete event systems. Proceedings of the IEEE 77(1), 81–98 (1989)
6. Queiroz, M.H., Cury, J.E.R.: Modular supervisory control of large scale discrete event systems. In: Boel, R., Stremersch, G. (eds.) WODES 2000. Discrete Event Systems: Analysis and Control, pp. 103–110. Kluwer Academic Publishers (2000)
7. Aalst, W., Hee, K., Werf, J., Kumar, A., Verdonk, M.: Conceptual model for on line auditing. Decision Support Systems 50(3), 636–647 (2011)
8. Schaidt, S., Vieira, A.D., Loures, E.F.R., Santos, E.A.P.: Dealing with Constraint-Based Processes: Declare and Supervisory Control Theory. In: Rocha, Á., Correia, A.M., Wilson, T., Stroetmann, K.A. (eds.) Advances in Information Systems and Technologies. AISC, vol. 206, pp. 227–236. Springer, Heidelberg (2013)
9. Pesic, M.: Constraint-based workflow management systems: Shifting control to users. Phd thesis, Eindhoven University of Technology, Eindhoven (2008)
10. Su, R., Wonham, W.M.: Supervisor reduction for discrete-event systems. Discrete Event Dynamic Systems 14, 31–53 (2004)
11. Rychkova, I., Nurcan, S.: Towards Adaptability and Control for Knowledge-Intensive Business Processes: Declarative Configurable Process Specifications. In: 44th Annual Hawaii International Conference on System Sciences, pp. 1–10. IEEE (2011)
12. Rychkova, I., Regev, G., Wegmann, A.: High-Level Design and Analysis of Business Processes the Advantages of Declarative Specifications. In: 2th IEEE International Conference on Research Challenges in Information Science, pp. 3–6. Marrakech (2008)
13. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Information Systems 30(5), 349–378 (2005)