Computer Science and Information Systems 11(4):1229–1247 DOI: 10.2298/CSIS131029026S

Modeling Constraint-based Processes: a Supervisory Control Theory Application

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

Pontificia Universidade Catolica do Parana, Graduate Program in Industrial and Systems Engineering and Graduate Program in Health Technology
Imaculada Conceicao 1155
Curitiba, Brazil
{eduardo.portela,agnelo.vieira, sauro.schaidt, eduardo.loures}@pucpr.br

Abstract. Constraint-based processes require a set of rules that limit their behavior to certain boundaries. In these processes, the control flow is defined implicitly as a set of constraints or rules, and all possibilities that do not violate any of the given constraints are allowed to be executed. 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 activities such that constraints are always obeyed. We demonstrate that our approach may be used as a declarative language for constraint-based processes. In order to provide support for users of such processes and to facilitate the using of our control approach, we offer a set of constraints modeled by automata. This set encompasses the constraints frequently needed in workflow system.

Keywords: constraint-based processes, Supervisory Control Theory, declarative languages, flexible processes.

1. Introduction

Nowadays constraint-based processes approaches have received increased interest [16]. In these processes, the control flow is defined implicitly as a set of constraints or rules, and all possibilities that do not violate any of the given constraints are allowed to be executed [13] [9] [8]. A constraint-based process model specifies the activities that must be performed to produce the expected results but it does not define exactly how these activities should be performed [6] [7]. Thus, any execution order of activities is possible provided that the constraints are not violated. Thus, most of time the process execution is driven by users choice.

DECLARE [13] [4] is developed as a constraint-based system and it uses a declarative language grounded in Linear Temporal Logic (LTL). DECLARE provides a graphical representation of constraints (DecSerFlow) [12] that hides the associated LTL formulas from users. According to [9] [8], this approach suffers from the fact that the subsequent tools for execution and analysis will refer to the LTL expression and not to the graphical notation. The full generality of LTL may lead to a poor execution time. For verification and enactment purposes, it is necessary to translate LTL to finite automata. While computers are very good at handling nite automata, the translation itself is often a roadblock as it may take time exponential in the size of the LTL formulas [24]. This motivates researching the problem of finding an expressive constraint-based processes approach where both the constraints as well as the run time state can be easily visualized and understood by the end user and also allows an effective verification (blocking,conflict, dead tasks) and execution of activities.

In the present paper we propose a new approach to deal with constraint-based processes founded on the Supervisory Control Theory [15]. The new approach proposes a control system which restrains the process in order to not violate the constraints. This action is accomplished through dynamic disabling of some events, restraining the state space of process. We consider that a process may contain sequences of events that are not allowed to occur. These sequences may violate a desired ordering of events and they need to be avoided. Thus, a supervisor is built in order to ensure that the whole set of constraints is not violated. We had some challenges bringing the formal foundation of SCT into a constraint-based process model, which characterizes the originality of our paper. We highlight the following contributions:

1. A new approach to deal with constraint-based processes. The proposed approach is based on SCT. The supervisor obtained applying SCT monitors and restricts execution of sequences of activities such that constraints are always obeyed. We demonstrate that our proposal can be used as a declarative language for constraint-based processes. Our approach does not limit the user by imposing rigid control-flow structures. In fact, the basis of our approach is to inform users of which activities are not allowed to occur after an observed trace of events at run-time, and users operate with some freedom because they choose execution sequences allowed under supervision;
2. A new approach to audit processes. Applying SCT results in a language (sequence of events) that considers all possible sequences which do not violate any of the constraints imposed to the process. It is possible to audit an execution of a process comparing if the performed sequence of activities belongs to that language.
3. Modeling activities and constraints using automata. We represent activities and constraints frequently needed in workflow systems using automata. This is necessary to apply the SCT. We propose a general model of activities as well as a set of constraints (Fig. 8 to 11, section 4). We aim to support users without a deep knowledge in SCT on its application in order to model constraint-based processes;

The present paper is an extended version of the previous paper presented in WorldCist 2013 [19], and is organized as follows: Section 2 describes the Supervisory Control Theory, as the fundamental concept of the proposed approach. Section 3 explains the modeling of activities using automata. Section 4 explains the modeling of constraints using automata. Also, it is presented four categories of constraints usually needed in business processes. Section 5 presents an application example to illustrate our approach. Section 6 discusses the process execution and the architecture of the run-time environment. Section 7 concludes the paper.

2. Supervisory Control Theory

Supervisory Control Theory (SCT)[15] has been developed in recent decades as an expressive framework for the synthesis of control for Discrete-Event systems (DES). According to SCT, the behaviour of a DES may be represented by sequences of events corresponding to ordered execution of activities. Among all possible sequences of events and due to the process rules and constraints, some sequences of events are desirable while other sequences are not since they violate these rules or constraints. Instead of defining a priori a specific sequence of events to be enforced in order to satisfy the constraints, the core concept of SCT is to design a supervisor that, following the sequence of events while the process evolves, specifies which events cannot occur in order to not violate the constraints. Thus, after the occurrence of an event the system (or the DES) decides which event will occur among those that are not disabled by a supervisor. SCT provides algorithms that, based on a process model considering all feasible event sequences and the associated constraints, allow one to design a supervisor whose control action imposes a minimally restrictive behaviour over a DES under consideration.

SCT is based on automata and formal language theories. Usually, a composed system is represented by a set of automata as $\{G_i | i \in I\}$, where $i \in I$ identifies each subsystem. Automaton G_i represents the independent behaviour of a corresponding subsystem in a high degree of abstraction. The uncoordinated or unconstrained behaviour of the entire DES is obtained by the synchronous product [2] of all subsystems as $G = ||_{\forall i \in I} Gi$. An automaton (also known as a language generator) is a structure as $G_i = (\Sigma^{Gi}, Q^{Gi}, \delta^{Gi}, q_0^{Gi}, Q_m^{Gi})$ where Σ^{Gi} is the alphabet (set) of events, Q^{Gi} is the set of states, $\delta^{Gi} : (Q^{Gi} x \Sigma^{Gi}) \rightarrow Q^{Gi}$ is the state transition function (in general, partially defined), q_0^{Gi} is the initial state, and $Q_m^{Gi} \subseteq Q^{Gi}$ is the set of marker states. An automaton state represents that a certain activity is being performed or that the subsystem is idle. Events represent the beginning and (un)successful execution of such activity. One may differentiate some states to give them a special meaning by grouping them in a set of marked states. In SCT marked states are those representing accomplishment of activities.

A Product System Representation (PSR) is a set of asynchronous subsystems such that all pairs of subsystems in $\{G_i | i \in I\}$ have disjoint alphabets. The system's whole set of events is $\Sigma = \cup_{\forall i \in I} \Sigma^{Gi}$. There are two languages associated with automaton G: the closed language $L(G)$ and the marked language $L_m(G)$. The closed language is the set of all sequences of events leading from the initial state to some state of G. The marked language is the set of all sequences of events leading from the initial state to any marked states such that $L_m(G) \subseteq L(G)$. These are the languages representing the unconstrained behaviour of the entire system. Under these languages there are several undesirable sequences of events that must be avoided in order to restrain the system inside a desirable (allowed) behaviour.

SCT allows the designer to take into account the nature of events. While there are some events whose occurrence might be disabled by a control agent there are events whose occurrence cannot be disabled. An event is controllable if a control agent (supervisor) can disable its occurrence. One may consider that a certain event is uncontrollable by convenience in order to not allow it to be disabled. In general, an uncontrollable event is inherently unpreventable. Considering a subsystem in $\{G_i | i \in I\}$, Σc^{Gi} denotes its set of controllable events and Σuc^{Gi} its set of uncontrollable events. The whole set of controllable events is $\Sigma c = \bigcup_{\forall i \in I} \Sigma c^{Gi}$.

Usually there is a set of constraints to be imposed to the system to restrain its uncoordinated behaviour. Each constraint may be represented by an automaton resulting in a set as $\{C_j | j \in J\}$, where $j \in J$ identifies each constraint. Performing the synchronous product of all automata in $\{C_j|j \in J\}$ with automaton G results automaton C representing a global constraint.

A supervisor is a map from the closed language of G to a subset of events to be enabled $S : L(G) \rightarrow 2^{\Sigma}$. A supervisor may be represented by an automaton and an output map $\{\Upsilon\} = (S, \Phi)$, where $S = (\Sigma^S, Q^S, \delta^S, q_0^S, Q_m^S)$. Automaton S is driven by occurrence of events in DES, and output map $\Phi : Q^S \rightarrow 2^{\Sigma_c}$ specifies the subset of controllable events that must be disabled as a correspondence of the active state of automaton S. The action of a supervisor includes disabling controllable events and unmarking sequences of events. Algorithms provided by SCT allow the formal synthesis of automaton Υ/G, which represents the optimal behaviour of G under supervision of Υ, where $L(\Upsilon/G) \subseteq L(G)$ and $L_m(\Upsilon/G) \subseteq (L(\Upsilon/G) \cap L_m(G))$. This behaviour is named supremal controllable sublanguage of $L_m(C)$ with regard to G and it is usually represented as $supC(E, G)$. Whenever $L_m(\Upsilon/G)$ is a proper subset of $(L(\Upsilon/G) \cap L_m(G))$, Υ is a marker supervisor, i.e., there are sequences of events corresponding to accomplished tasks in the uncoordinated behaviour of G that are no longer considered accomplished tasks under the action of the supervisor. Typically, the automaton representing a supervisor is automaton Υ/G itself. [10] and [21] provide algorithms to obtain a reduced representation of supervisor Υ as a new pair (S_r, Φ_r) where automaton S_r has a smaller number of states than Υ/G and it provides the same control action.

In a monolithic approach, a single global supervisor is synthesised to cope with all constraints. Necessary and sufficient conditions for the existence of supervisors are established in [14]. According to Local Modular Control (LMC) [3], an extension of the SCT, instead of synthesizing a single global supervisor that satisfies the entire set of constraints, a local supervisor must be synthesized for each constraint in $\{C_j|j \in J\}$. This leads to a set of local supervisors $\{\Upsilon_j|j \in J\}$. Synthesis of local supervisor Υ_j is performed considering corresponding local constraint Cl_j and its corresponding local plant Gl_j. A local plant is obtained performing the synchronous product of only those subsystems in $\{G_i|i \in I\}$ sharing some event with corresponding constraint and local constraint is obtained performing the synchronous product of automata representing corresponding constraint and local plant $(Cl_j = Gl_j||C_j)$. Automaton Υ/Gl_j represents the optimal behaviour of local plant Gl_j under supervision of corresponding local supervisor Υ_j. If at least one local supervisor in set $\{\Upsilon_j|j \in J\}$ disables an event, the event is disabled in G. A sequence of events is recognized as an accomplished task if all local supervisors agree with.

A limitation of LMC is that the behaviour obtained under the action of all local supervisors may fail to be non-blocking, even if each modular supervisor is non-blocking. Blocking in SCT occurs when all possible ways of continuing a sequence of events never lead to a marked state. After synthesis of all local supervisors, it is necessary to verify whether their control actions are free of conflicts. One way is confirming $\Upsilon/G = ||\forall j \in J \Upsilon/Gl_j$. In the worst case, such verification involves the same complexity as that found during synthesis of the global supervisor [3]. If this property is verified, the behaviour obtained under the action of the entire set of local supervisors is identical to the behaviour obtained under the action of a global supervisor.[25] proposes how to proceed if such property is not verified.

We believe that the Supervisory Control Theory (SCT) is a promising candidate for modeling and execution of a constraint-based process. We highlight the following reasons:

- SCT uses automata as formalism to represent activities, constraints and the resulting supervisor. This is a formal and explicit way of representing them;
- Using SCT, from modelling to synthesis and visualization, the formal notation is always the same (automata), without the need to convert from one notation to another (in DECLARE is necessary to convert LTL formulas to automata);
- In SCT the state of each activity as well as each constraint may be easily visualized and understood by the end user at run-time;
- SCT provides algorithms to perform a formal synthesis of supervisor (or the admissible language of a constraint based process) instead of the usual manual and heuristic procedures;
- The obtained solution is minimally restrained and also dead-lock free;
- New control actions may be rapidly and formally designed when modifications, such as redefinition of constraints or activities 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 (concern to the admissible language of the process). This is a very strong characteristic of the proposed approach. As far as we know, there is no approach that offer a better solution related to the admissible language.

3. Modeling activities of constraint-based processes

The theory presented at previous section, together with a method of control implementation [23], have successfully been employed on the actual control of DES with characteristics of the manufacturing industry [20] [5]. In order to provide full support to control implementation in the context of constrain-based processes, we propose first to analyze the modeling of activities and constraints. This section presents how we propose to represent activities of constraint based processes so that they may be coordinated by supervisors obtained applying SCT.

Suppose a process with a set of associated activities $A = \{a_i | i \in I\}$ where I is a set of index uniquely identifying each activity. We propose that each activity is modelled as a corresponding automaton $A_i = (\Sigma^{Ai}, Q^{Ai}, \delta^{Ai}, q_0^{Ai}, Q_m^{Ai})$, as shown in Fig. 1(a), resulting in set $\{A_i | i \in I\}$. States on this automaton mean that activity is being performed (state 1) or is not being performed (state 0). Transition from state 0 to state 1 is due to event *start activity ai* (*si*); transition from state 1 to state 0 is due to occurrence of event *successfully complete activity ai* (*ci*) or *cancel activity ai* (*xi*). In SCT marked states are those representing accomplishment of tasks, a state represented with a double line is a marked state. In this model *si* is a controllable event while *ci* and *xi* are uncontrollable events. It means that starting an activity by a resource may be disabled by a supervisor. However, once it is under execution a supervisor is not allowed to avoid it to be *cancelled* or *completed successfully*. The set of events of automaton A_i is $\Sigma^{A_i} = \{si, ci, xi\}$. Considering the entire set of activities, the whole set of events is $\Sigma = \bigcup_{\forall i \in I} \Sigma^{Ai}$. It can be seen that all pairs of automata in $\{Ai | i \in I\}$ have disjoint alphabets of events ($\forall p, q \in I, \Sigma^{Ap} \cap \Sigma^{Aq} = \emptyset$), so this is a product system representation. If desired, a more detailed automaton may be employed, including more states and events, it is also possible to apply a different interpretation of events' controllability. For example, it is possible to consider the activity life cycle as stated in [22] and [16]. The corresponding automaton may include states as *allocated*, *suspended*, *failed*, and events as *suspend*, *resume*, *allocate*, as shown in [17] and

[18]. The modeler has to choose which states and events will be considered based on the relevant constraints to be imposed to the process under consideration.

Considering the whole set of activities, it is possible to have several activities being executed at the same time. Performing synchronous product of all automata in $\{A_i|i \in I\}$, it results on automaton A where the set of states represents all possible combinations of activities being performed over a certain process instance. It is a subset of the cartesian product of the set of states of all automata in $\{A_i|i \in I\}$. Since this set of automata is a product system representation the number of states of A is 2^n, where n is the number of automata. Automaton A represents the uncoordinated behaviour of activities. In automaton A an state is marked if and only if it corresponds to a combination of marked states of automata in $\{A_i|i \in I\}$. For instance, considering a process with only two activities $(A = \{a1, a2\})$, the synchronous product of corresponding automata $(A = A1||A2)$ is the one shown in Fig. 1. At this automaton each state is named as an ordered pair (state of $A1$, state of $A2$).

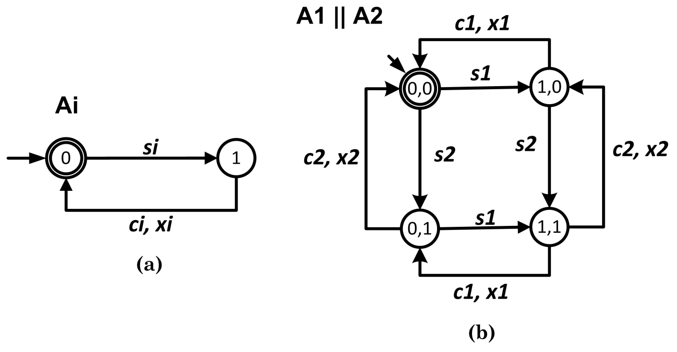


Fig. 1. Automata representing: (a)activity model and (b) uncoordinate behaviour model of two activities

4. Modeling constraints

According to [1], three main categories of constraints may be identified. One category focuses on ensuring that each process instance is performed under specific ordering of activities. The second category focuses on managing the allocation or usage of resources that perform such activities. The third one focuses on the attributes of a process instance. SCT may be employed to synthesize supervisors to enforce these constraints. This paper is restricted on modelling activities compatible with constraints on the first category. In this section we describe the procedure to represent a constraint using automata.

Consider automaton **A** representing the uncoordinated behaviour of a set of activities $\{ai|i \in I\}$ of a process. Language $L(A)$ represents all sequences of events that may be performed by these activities without any constraint, and $L_m(A)$ is a subset of $L(A)$ representing accomplished activities. The basic premise is that a process contains sequences of events in $L(A)$ that are not acceptable because they violate some constraint. It is also possible that certain states must be forbidden since they represent an unauthorized concurrent execution of activities. These sequences and states must be avoided. Also, it is possible that a sequence of events in $L_m(A)$ does not correspond to an accomplished task when the process instance is performed under supervision. Thus, that sequence needs to be unmarked by supervisor. Consider automaton S/A, such that $L_m(S/A) = supC(C, A)$, the one that recognizes the supremal controllable language of constraint activities. [16] define a supported traces as a sequence of events complying with all mandatory constrains. This definition complies with the definition of a sequence of events belonging to $L_m(S/A)$. Fig. 2 shows the languages relation: the region 1 includes sequences of events belonging to $L_m(S/A)$ (or supported traces); the region 2 includes sequences w such that $w \in L(A)$, $w \in L_m(A)$, $w \not\ni L(S/A)$, $w \not\ni L_m(S/A)$; the region 3 includes sequences $w \in L(A)$, $w \not\ni L_m(A)$, $w \in L(S/A)$, $w \not\ni L_m(S/A)$, the region 4 includes sequences $w \in L(A)$, $w \in L_m(A)$, $w \in L(S/A)$, $w \not\ni Lm(S/A)$.

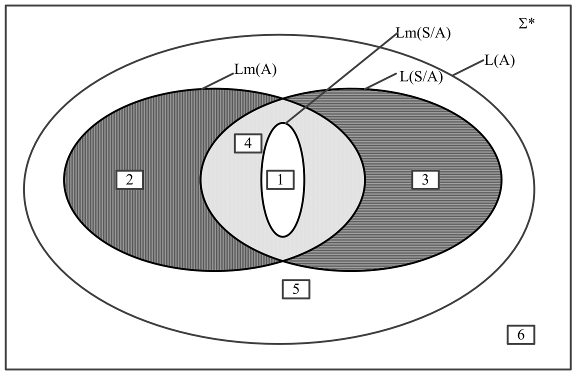


Fig. 2. Venn diagram of languages relation

To formally obtain the supervisor that restrains the uncoordinated behaviour of activities it is necessary to express constraints in terms of automata. Usually, each constraint is represented as an automaton resulting in set $\{C_j | j \in J\}$, where J is a set of index uniquely identifying each constraint. When a process instance is performed under supervision of a set of supervisors obtained employing SCT, the related constraints will never be violated and there will always be at least one sequence of events leading to a marked state, i.e., there will always be the possibility of accomplishing a task.

Modelling constraints is based on sequence of events and unmarking states. Consider an automaton $Cz \in \{Cj | j \in J\}$. Usually the alphabet of events of Cz is only a proper subset of the whole set of events. Such alphabet contains the events strictly necessary to represent the constraint and it is represented as Σ^{Cz}. If the occurrence of an event in Σ^{Cz} is not represented at a certain state of Cz, either in self-loop leading to the same state or to a different one, then the occurrence of this event will not be allowed after any sequence of events leading to such state. If a state in Cz is not a marked one then the sequences of events leading to it will not be considered as accomplished tasks, even if any of these sequences lead to marked states in another automaton in $\{Cj | j \in J\}$ or in automaton A.

The *existence(a1, 1)* model requires that *activity a1 must occur at least once at every trace* [16][11]. In order to facilitate the understanding of our approach, we rewrite this constraint to *activity a1 is successfully completed at least once*. Fig. 3 presents two possibilities of modelling this constraint. The first possibility is through automaton **C1**. Initial state of **C1** has a self-loop labelled as $\Sigma - c1$, meaning that the occurrence of any event belonging to Σ but $c1$ keeps this state as the active one. The active state is state 1 only after occurrence of event $c1$. This remains the active state despite the occurrence of any event, as there is a self-loop labelled as Σ. Since the only marked stated is state 1 then accomplishing a task is recognized only after first occurrence of event $c1$. Alphabet of events of this automaton is the whole set of events $\Sigma^{C1} = \Sigma$. Modelling a constraint through an automaton whose alphabet of events is Σ has the advantage of clearly presenting the occurrence of all possible events. In the case one employs this automaton as a constraint under the LMC approach, then the corresponding local plant is automaton A and does not take advantage of this approach in reducing computational complexity to synthesize a corresponding local supervisor. Considering definition of synchronous product and algorithms for the synthesis of supremal controllable language $supC((A\|C1), A)$ [26], a more efficient representation of a constraint is through an automaton employing only strictly necessary events.

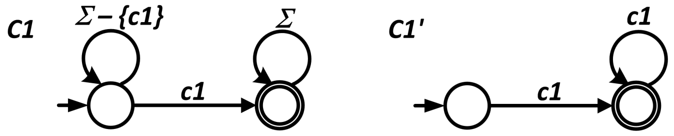


Fig. 3. Automata representing constraint *existence(a1, 1)*

The second possibility of representing *existence(a1, 1)* model is through automaton **C1'**, where the alphabet of events is $\Sigma^{C1'} = c1$. In this case it is implicit that any event that does not belong to $\Sigma^{C1'}$ is always allowed to occur in accordance with automaton representing the uncoordinated behaviour as automaton **A**. Adopting **C1'** as a constraint, results that corresponding local plant is automaton **A1** alone. Thus computational complexity on the synthesis of local supervisor and number of states of automaton representing it will be smaller than in the first possibility. Fig. 4 presents automata $Sc1/A$ and $Sc1'/A1$, where $L_m(Sc1/A) = supC((A\|C1), A)$ and $L_m((Sc1'/A1) = supC((A1\|C1'), A1)$ may be employed as supervisors enforcing constraint *existence(a1, 1)* over the set of activities $A = \{a1, a2\}$. Control action of supervisors obtained in both possibilities will be equivalent. Such control action is only unmarking sequences of events (recognizing accomplished tasks), it will not disable events. Applying supervisor's reduction algorithms [26] on these supervisors results automaton $Sc1r$, also shown in Fig. 4. It can be seen from automaton **A** (Fig. 1) that sequences $w1 = \varepsilon$ (the empty sequence of events), $w2 = s1x1$, $w3 = s2c2$, $w4 = s1c1$, among others, lead from initial state back to it and this is a marked state so $w1$, $w2$, $w3$, $w4 \in L_m(A)$. It can also be seen from automaton $Sc1/A$ and $Sc1'/A1$ that while sequence $w4$ (region 1 according to Fig. 2) leads from initial state to a marked state the same is not true for sequences $w1$, $w2$ and $w3$; this means that $w4 \in L_m(Sc1/A)$ but $w1$, $w2$, $w3 \not\ni L_m(Sc1/A)$ (region 4 according to Fig. 2). While $w4$ is a supported trace, as defined by [16], $w1$, $w2$ and $w3$ are unsupported ones. Language $L_m(Sc1/A)$ contains all possible sequences of events recognized as accomplished tasks under supervision. Thus, it contains all supported traces.

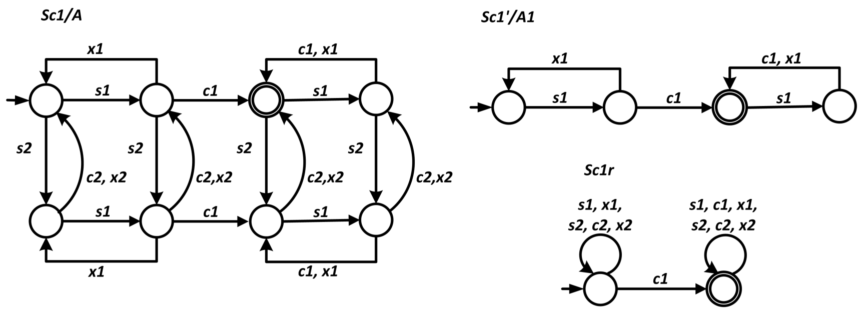


Fig. 4. Automata Sc1/A, Sc1'/A1 and Sc1r

According to [16] and [11], the constraint *response(a1, a2) states that if a1 is executed, a2 needs to be executed afterwards (but not directly after).* We may rewrite it replacing *executed* by *completed*. Fig. 5 shows the automaton **C2** as a possible model for representing this constraint employing only strictly necessary events, where $\Sigma^{C2} = \{c1, c2\}$. In this automaton the state transition function is defined with the occurrence of all events in Σ^{C2} at every state meaning that they are always allowed to occur. While state 0 is a marked one, state 1 is not, meaning that sequences of events leading to state 1 are not considered as accomplished tasks because there has been at least one occurrence of $c1$ that was not followed by $c2$. In this case automaton **A** represents the local plant since that constraint employs events of every activities. Automaton $Sc2/A$, where $L_m(Sc2/A) = supC((A\|C2, A)$, is a possible supervisor's representation of a supervisor enforcing this constraint, and automaton **Sc2r** is a reduced representation of it. Again, supervisor's control action is only unmarking sequences of events: the output map is always empty, i.e. $\forall q \in Q^{Sc2/A} \rightarrow (\Phi(q) = \emptyset)$. As shown in [16], $< a1, a2 >$, $< a1, a1, a1, a2 >$ and $< a2 >$ are supported traces while $< a1 >$ is an unsupported trace. In these cases it is only considered an abstraction of activities, i.e., only considered a single event representing the execution and completion of an activity. Also, it is not considered the overlapping of activities, i.e., activities are only sequentially executed. Sequences of events that may represent traces with *start* and *complete* events, may be $w5 = s1c1s2c2$, $w6 = s1c1s1c1s1c1s2c2$, $w7 = s2c2$, $w8 = s1c1$. Considering trace $< a1, a2 >$ there are many other sequences of events, including overlapping activities. For instance $w9 = s1x1s1x1s1c1s2c2$, $w10 = s1s2c1c2$, $w11 = s1c1s2x2s2x2s2c2$, all traces belonging to $L_m(Sc2/A)$. While $w8$ is an unsupported trace (region 4 in Fig. 2), $w5$, $w6$, $w7$, $w9$, $w10$ and w11 are supported traces (region 1 in Fig. 2).

[16] and [11] also present the constraint *precedence(a1, a2)* as *activity a2 needs to be preceded by activity a1*. We may rewrite it as *a2 can be completed only after a1 has been completed at least once*. Fig. 6 shows the automaton **C3** as a possible model for representing this constraint. Notice that both states are marked, meaning that corre-

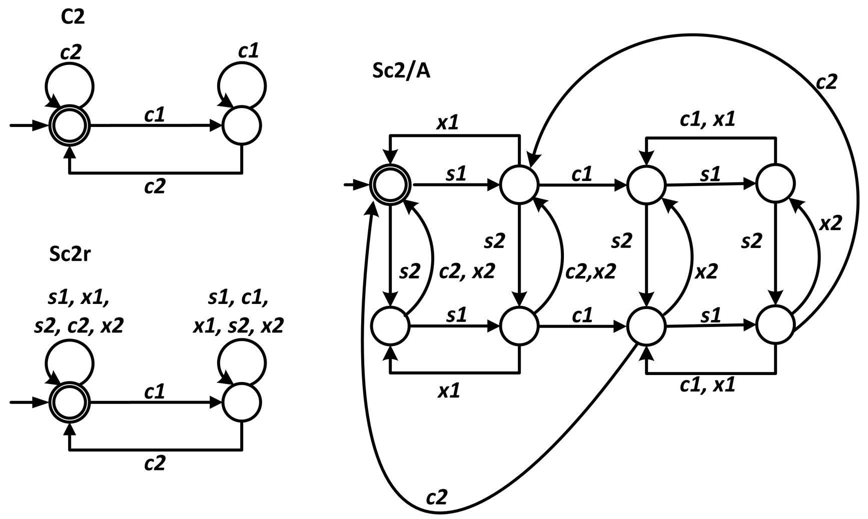


Fig. 5. Automata C2, Sc2/A, and Sc2r

sponding supervisor will not unmark sequence of events. Since $\Sigma^{C3} = \{c1, c2\}$ and state transition function is not defined with the occurrence of $c2$ at state 0 than this event cannot occur at this state, i.e. prior to first occurrence of $c1$. Automaton $Sc3/A$, where $L_m(Sc3/A) = supC((A \parallel C3), A)$, is a possible supervisor's representation. In this case supervisor's control action is only disabling controllable events, and corresponding output map specifies that event $s2$ is disabled at states 0 and 1 ($\Phi(0) = \Phi(1) = \{s2\}, (\forall q \in Q^{Sc3/A}, q \neq 0, q \neq 1) \to (\Phi(q) = \emptyset)$). The aim of this supervisor is to avoid occurrence of event $c2$ prior to the first occurrence of $c1$. Since $c2$ is considered to be an uncontrollable event then the supervisor needs to take an anticipatory action disabling $s2$ (a controllable event).

Fig. 7 presents automata $Sc3/A'$ and $Sc3r'$ considering $c2$ as a controllable event. Notice that automaton $Sc3/A'$ has two extra states due to occurrence of event $s2$ from state 0 and from state 1. Also, supervisor's control action is disabling occurrence of event $c2$ at these extra states (6 and 7) instead of event $s2$ at states 0 and 1. As shown in [16], $< a1, a2 >$, $< a1, a2, a2, a2 >$ and $< a1 >$ are supported traces while $< a2 >$ is an unsupported trace. Sequences of events that may represent such traces are, respectively, $w12 = s1c1s2c2$, $w13 = s1c1s2c2s2c2s2c2$, $w14 = s1c1$, $w15 = s2c2$. While $w15$ is an unsupported trace (region 2 in Fig. 2), $w12$, $w13$, $w14$ are supported traces (region 1 in Fig. 2). Considering constraint precedence $(a1, a2)$ and $c2$ as an uncontrollable event, sequences in region 3 in Fig. 2 are $w16 = s1$ and $w17 = s1c1s2$; a sequence in region 5 is $w18 = s2$ and a sequence in region 6 is $w19 = c1s1$.

According to [16], constraint-based process models focus on what should be done by describing the activities that may be performed and the constraints prohibiting undesired execution behaviour. In the present paper we restrict our focus on constraints aiming to ensure that each process instance is performed under an ordering of activities and we use the same principle as proposed in [11] and [19], where is considered four groups

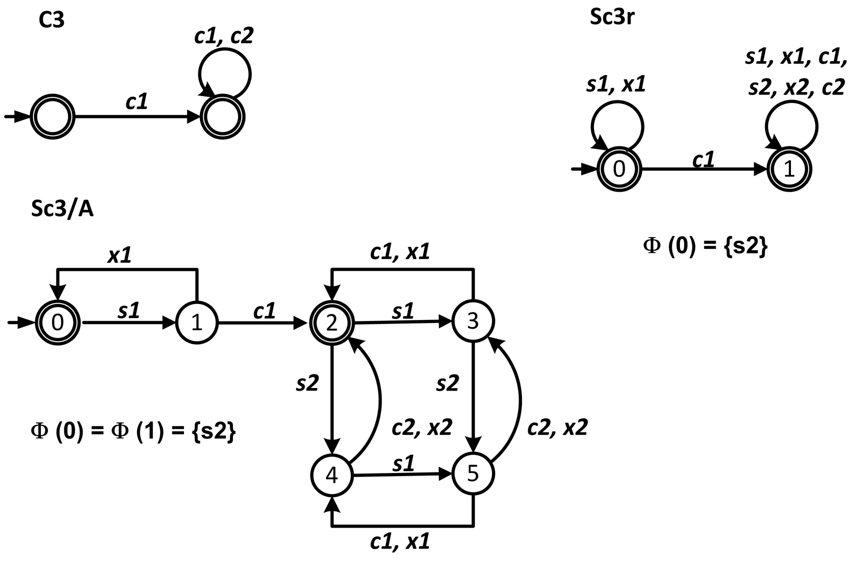


Fig. 6. Automata C3, Sc3/A, and Sc3r with c2 as an uncontrollable event

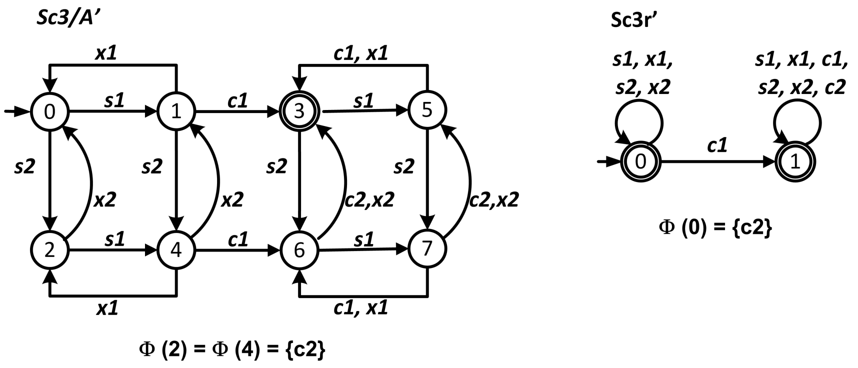


Fig. 7. Automata Sc3/A', and Sc3r' with c2 as a controllable event

of constraints: (1) existence, (2) relation, (3) negation and (4) choice. Existence models specify how many times or when one activity may 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. Because the space limitation of this paper, we only present some models of each group. Fig. 8 to Fig. 11 shows some constraint models using automata.

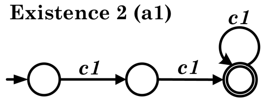
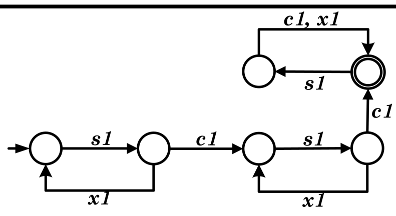
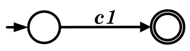
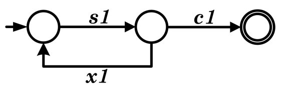
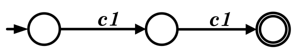
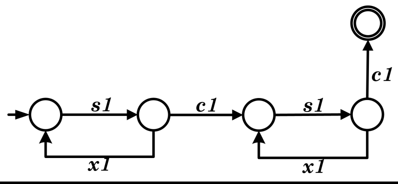
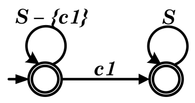
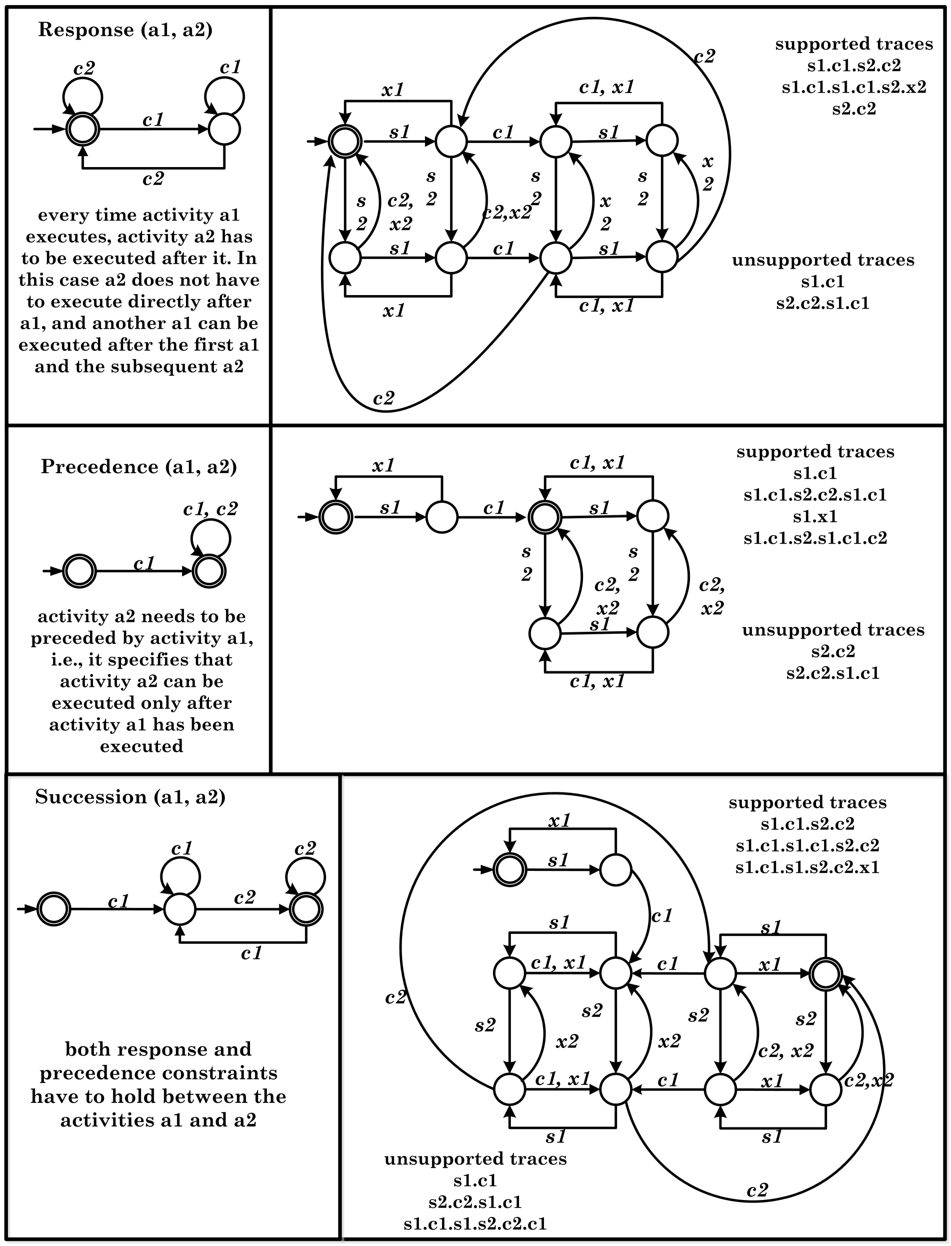
Existence 2 (a1)  activity a1 is executed (completed) at least twice	 supported traces s1.c1.s1.c1 s1.c1.s1.c1.s1.c1 s1.c1.s1.c1.s1.x1 unsupported traces s1.x1 s1.c1 s1.c1.s1.x1
Exactly 1 (a1)  activity a1 is executed (completed) exactly once	 supported traces s1.c1 unsupported traces s1.x1
Exactly 2 (a1)  activity a1 is executed (completed) exactly twice	 supported traces s1.c1.s1.c1 unsupported traces s1.x1 s1.c1 s1.c1.s1.x1
Init (a1, a2, ...an)  activity a1 must be the first executed activity in an instance	supported traces s1.c1.(S)* s1.x1.s1.c1.(S)* unsupported traces (S)*.s1.c1

Fig. 8. Automata representing the group existence

5. Application example

Project management usually consists of various management processes, monitoring and control activities. These processes are performed in different conditions for each new project, which requires a flexible modeling. One of the most popular benchmarks for project management is the PMBOK (Project Management Body of Knowledge). PMBOK in its fourth version establishes a set of 42 macro-processes in nine knowledge areas. The *Collect Requirements* process was selected for the implementation of declarative model-



Response (a1, a2)

every time activity a1 executes, activity a2 has to be executed after it. In this case a2 does not have to execute directly after a1, and another a1 can be executed after the first a1 and the subsequent a2

supported traces
s1.c1.s2.c2
s1.c1.s1.c1.s2.x2
s2.c2

unsupported traces
s1.c1
s2.c2.s1.c1

Precedence (a1, a2)

activity a2 needs to be preceded by activity a1, i.e., it specifies that activity a2 can be executed only after activity a1 has been executed

supported traces
s1.c1
s1.c1.s2.c2.s1.c1
s1.x1
s1.c1.s2.s1.c1.c2

unsupported traces
s2.c2
s2.c2.s1.c1

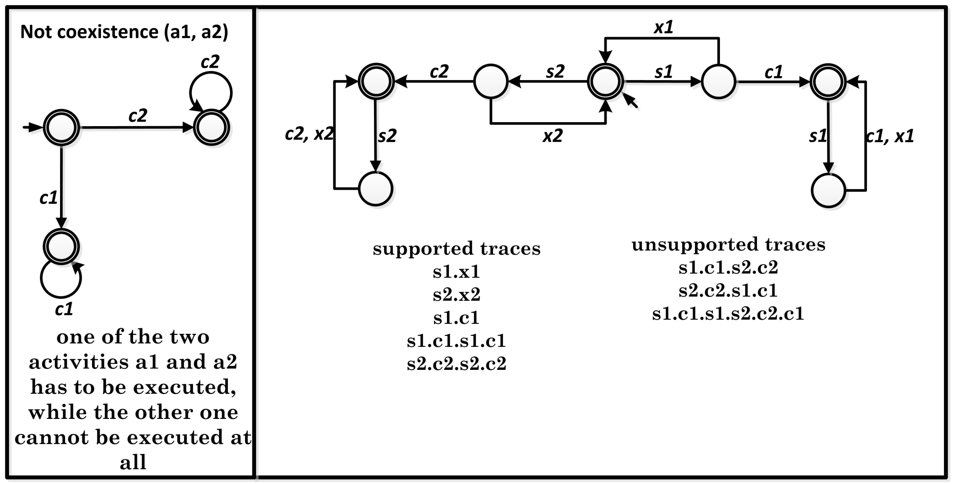
Succession (a1, a2)

both response and precedence constraints have to hold between the activities a1 and a2

supported traces
s1.c1.s2.c2
s1.c1.s1.c1.s2.c2
s1.c1.s1.s2.c2.x1

unsupported traces
s1.c1
s2.c2.s1.c1
s1.c1.s1.s2.c2.c1

Fig. 9. Automata representing the group relation

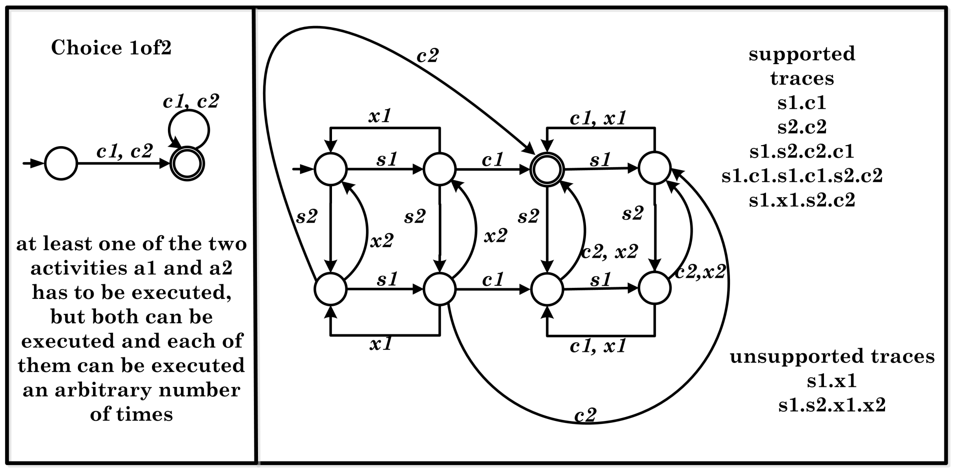


one of the two activities a1 and a2 has to be executed, while the other one cannot be executed at all

supported traces
s1.x1
s2.x2
s1.c1
s1.c1.s1.c1
s2.c2.s2.c2

unsupported traces
s1.c1.s2.c2
s2.c2.s1.c1
s1.c1.s1.s2.c2.c1

Fig. 10. Automata representing the group negation



at least one of the two activities a1 and a2 has to be executed, but both can be executed and each of them can be executed an arbitrary number of times

supported traces
s1.c1
s2.c2
s1.s2.c2.c1
s1.c1.s1.c1.s2.c2
s1.x1.s2.c2

unsupported traces
s1.x1
s1.s2.x1.x2

Fig. 11. Automata representing the group choice

ing techniques and illustration of the approaches presented here. The goal of this process is to identify the set of requirements of the final product of a project.

The PMBOK provides three stages for each process: Inputs, Tools and Techniques, and Outputs. The inputs to this process are the documents *Project Charter* (PC) and *Stakeholder Register* (SR). The tools and techniques adopted for implementing this model are: interviews, focus groups, facilitated workshops, questionnaires and surveys, prototypes and brainstorm. The outputs suggested by PMBOK are *Requirements Documentation* (RD), *Requirements Management Plan* (RMP) and *Requirements Traceability Matrix* (RTM). For this work we selected the output *Documentation Requirements*. Thus, activities under control are *Review Project Charter and Stakeholders Register* ($a1$), *Brainstorm* ($a2$), *Focus Groups* ($a3$), *Facilitated Workshops* ($a4$), *Questionnaires and Surveys* ($a5$), *Interviews* ($a6$), *Prototypes* ($a7$), and *Requirements Documentation* ($a8$). We assume each activity ai (i=1,,8) is modeled as an automaton shown in Fig. 1 (a).

There are five constraints specified for this process: constraint $C1$ defines that *Review Project Charter and Stakeholders Register* ($a1$) must be the first executed activity in an instance; constraint $C2$ defines that at least one of the five activities *Brainstorm* ($a2$), *Focus Groups* ($a3$), *Facilitated Workshops* ($a4$), *Questionnaires and Surveys* ($a5$) and *Interviews* ($a6$) has to be executed, but all can be executed and each of them can be executed an arbitrary number of times; constraint $C3$ defines that activities *Focus Groups* ($a3$) and *Facilitated Workshops* ($a4$) have a not coexistence relation - only one can occur in every trace; constraint $C4$ defines that activity *Prototype* ($a7$) needs to be preceded by activity *Interviews* ($a6$); constraint $C5$ defines that activity *Requirements Documentation* ($a8$) is executed at least once.

6. Executing the supervisory control

Once an executable process model has been deployed to a run-time environment, new process instances can be created and executed according to this model. Generally, several instances of the same process model may exist representing different business cases (e.g., projects of different products). Our proposal is that the supervisory control coordinates the concurrent execution of these process instances.

When the preconditions for executing a particular activity are met during run-time, a new instance of this activity is created. Hence, an activity instance represents a single invocation of an activity during the execution of a particular process instance. Particularly, when a human activity becomes enabled during the execution of a process instance, the Process-Aware Information System (PAIS) first determines all resources qualifying for this activity instance. For each potential resource, a work item referring to the activity instance is created and added to his worklist. Work items related to a particular activity instance may be added to different user worklists. Generally, a worklist comprises all work items currently offered to, or processed by, a user [16].

Generally, resources (process participants or users) interact with a PAIS via worklists. When a resource allocates a work item from his worklist, all work items related to the same activity instance are removed from the worklists of other resources. Further, the resource to whom the work item is allocated may then trigger the start of the application service associated with the corresponding activity instance. The supervisory control must ensure that activities are executed considering the specified constraints during run-time

[16]. Fig. 12 shows three process instances PI_1, PI_2 and PI_3 running on the process presented in Section 5. The depicted worklists of resources John, Paul and George comprises work items related to these three process instances.

- Process instance PI_1: Activities $a1$, $a2$, $a3$ and $a6$ have already been completed but they are still enabled. This is because they may be executed any number of times. Notice that these activities have been added to the worklists of John and Paul. Activity $a4$ is disabled by the supervisory control and the activities $a5$,$a7$ and $a8$ are enabled and they have not been executed yet. These activities have been added to the worklists of John and George.
- Process instance PI_2: Activities $a1$, $a2$ and $a4$ have already been completed but they are still enabled. This is because they may be executed any number of times. Notice that these activities have been added to the worklists of John and Paul. Activities $a3$ and $a7$ are disabled by the supervisory control and the activities $a5$,$a6$ and $a8$ are enabled and they have not been executed yet. These activities have been added to the worklists of Paul and George.
- Process instance I_3: Only activity $a1$ is enabled (and it has not been executed yet). Activities $a2$ to $a8$ are disabled by the supervisory control. Notice that in this state the process has not been initiated. Activity $a1$ has been added to the worklist of John.

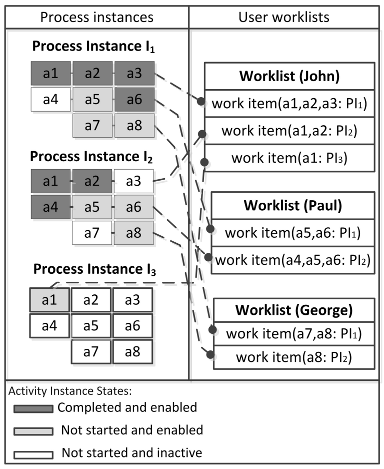


Fig. 12. Project management process, process instances and users worlists

According to our approach, during the execution of activities the supervisory control has to disable activities in order to not violate the constraints. In terms of run-time environment, the supervisory control cannot add an activity in any worklist if such activity is disabled by the supervisor. As long as an event (si, ci or xi) occurs, the state of the supervisor is updated and a new control action (a list of disabling activities) is established. Notice that in our approach the activities may be executed without overlapping (sequentially executed only) and with overlapping (executed in parallel). Fig. 13 illustrates the execution of the process shown in Section 5 considering overlapping activities for a specific process instance.

After creating a new process instance, only activity $a1$ is enabled, as the supervisor disables the others to not violate the constraint $C1$. With the completion of activity $a1$, the supervisor disables only the activity $a7$ (to not violate the constraint $C4$). This is followed by event $s2$ resulting in activity instance state $a2$ *running*. Next activity $a3$ is started resulting in activity instance state $a2$ and $a3$ *running*. Then activity $a2$ is completed. The set of disabled activities remains unaffected until the activity $a3$ has been completed. At this point the supervisor disables the activity $a4$, in order to not violate the constraint $C3$ (notice that the supervisor continues disabling the activity $a7$). After a while the activity $a6$ is started, followed by the beginning of activity $a8$, resulting in activity instance state $a6$ and $a8$ *running*. When the activity $a6$ is completed, the activity $a7$ becomes enabled. However, the process instance is finished when the activity $a8$ is completed.

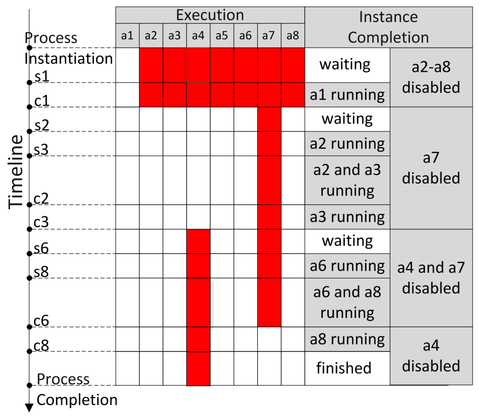


Fig. 13. Executing the project management process under supervision

7. Conclusion

We propose a new approach to deal with constraint-based processes. The proposed approach is based on Supervisory Control Theory, a formal foundation for building supervisors for DES. The supervisors proposed in this paper monitor and restrict execution sequences of activities such that constraints are always obeyed. We demonstrate that our proposal can be used as a declarative language for constraint-based processes. The proposed approach works informing users which activities are not allowed after an observed trace of events at run-time. Users can adopt this service as a guide to execute tasks with a guarantee that constraints are followed and goals are met. SCT allows a formal 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 a task).

References

1. Van der Aalst, W., Van Hee, K., Van der Werf, J.M., Kumar, A., Verdonk, M.: Conceptual model for online auditing. Decision Support Systems 50(3), 636–647 (2011)
2. Cassandras, C.G., Lafortune, S.: Introduction to discrete event systems. Springer (2008)
3. De Queiroz, M.H., Cury, J.E.: Modular supervisory control of large scale discrete event systems. In: Discrete Event Systems, pp. 103–110. Springer (2000)
4. van Der Aalst, W.M., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development 23(2), 99–113 (2009)
5. Diogo, R.A., Santos, E.A., Vieira, A.D., Loures, E.d.F., Busetti, M.A.: A computational control implementation environment for automated manufacturing systems. International Journal of Production Research 50(22), 6272–6287 (2012)
6. Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: Enterprise, Business-Process and Information Systems Modeling, pp. 353–366. Springer (2009)
7. Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: the issue of maintainability. In: Business Process Management Workshops. pp. 477–488. Springer (2010)
8. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Safe distribution of declarative processes. In: Software Engineering and Formal Methods, pp. 237–252. Springer (2011)
9. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Declarative modelling and safe distribution of healthcare workflows. In: Foundations of Health Informatics Engineering and Systems, pp. 39–56. Springer (2012)
10. Minhas, R.S.: Complexity reduction in discrete event systems. Ph.D. thesis, University of Toronto (2002)
11. Pesic, M.: Constraint-based workflow management systems: shifting control to users. Ph.D. thesis (2008)
12. Pesic, M., van der Aalst, W.M.: A declarative approach for flexible business processes management. In: Business Process Management Workshops, pp. 169–180 (2006)
13. Pesic, M., Schonenberg, H., van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International. pp. 287–287. IEEE (2007)
14. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM journal on control and optimization 25(1), 206–230 (1987)
15. Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE 77(1), 81–98 (1989)
16. Reichert, M.U., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer (2012)
17. Santos, E.A., Francisco, R., Vieira, A.D., de FR Loures, E., Busetti, M.A.: Modeling business rules for supervisory control of process-aware information systems. In: Business Process Management Workshops. pp. 447–458. Springer (2012)
18. Santos, E.P., Francisco, R., Pesic, M., van der Aalst, W.: Supervisory control service for supporting flexible processes. Industrial Management & Data Systems 113(7), 1007–1024 (2013)
19. Schaidt, S., Vieira, A.D., Loures, E.d.F.R., Santos, E.A.P.: Dealing with constraint-based processes: Declare and supervisory control theory. In: Advances in Information Systems and Technologies, pp. 227–236. Springer (2013)
20. Silva, D.B., Vieira, A.D., Loures, E.F., Busetti, M.A., Santos, E.A.: Dealing with routing in an automated manufacturing cell: a supervisory control theory application. International Journal of Production Research 49(16), 4979–4998 (2011)
21. Su, R., Wonham, W.M.: Supervisor reduction for discrete-event systems. Discrete Event Dynamic Systems 14(1), 31–53 (2004)

22. Ter Hofstede, A., Hofstede, A.H., Aalst, W.M., Adams, M.: Modern Business Process Automation: YAWL and its support environment. Springer (2010)
23. Vieira, A.D., Cury, J.E.R., de Queiroz, M.H.: A model for plc implementation of supervisory control of discrete event systems. In: Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on. pp. 225–232. IEEE (2006)
24. Westergaard, M.: Better algorithms for analyzing and enacting declarative workflow languages using ltl. In: Rinderle-Ma S., Toumani F., W.K. (ed.) Business Process Management LNCS, vol. 6896, pp. 83–98. Springer (2011)
25. Wong, K.C., Wonham, W.M.: Hierarchical control of discrete-event systems. Discrete Event Dynamic Systems 6(3), 241–273 (1996)
26. Wonham, W.M.: Design software: Xptct. Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto (2011)

Eduardo Alves Portela Santos is an Associate Professor at Pontificia Universidade Catolica do Parano (PUCPR), Curitiba, Brazil, where he chairs the Information System Group. His research interest include business process management, monitoring and control of business process, process modeling and analysis. In 2009 he was visiting researcher at the Department of Mathematics and Computer Science at the Eindhoven University of Technology, The Netherlands, under the supervision of Prof. Wil van der Aalst. Eduardo Alves Portela Santos is the corresponding author and can be contacted at: eduardo.portela@pucpr.br

Agnelo Denis Vieira is an Associate Professor at Pontificia Universidade Catolica do Parana - Health Technology Graduate Program, Curitiba, Brazil. He has earned degrees of Bachelor and Master in Mechanical Engineering as well as Doctor in Electrical Engineering from Federal University of Santa Catarina, Brasil. His research interests are on Modelling and Control of Discrete Event Systems applied at Health Care Systems and Protocols as well as on Experimental Analysis and Computer Simulation of Human Movement.

Sauro Schaidt earned his Bachelor of Computer Science degree from Federal University of Parana in 2004. He received his Master of Science degree in Industrial and Systems Engineering from the Pontifical Catholic University of Parana, Curitiba, Brazil, in 2013. Currently, he is PhD student at Graduate Program in Industrial and Systems Engineering. His research focus on supporting the execution of flexible processes by means of monitoring and control.

Eduardo de Freitas Rocha Loures is an Associate Professor at Pontificia Universidade Catolica do Parana (PUCPR) and an Associate Professor at the Federal University of Technology - Parana (UTFPR), both in Brazil. He is from 2010 the Education Chair of the International Society of Automation (ISA, District South America, Section Curitiba, Brazil). On 2012, he spent one year as a Visiting Academic at the Research Center for Automatic Control (CRAN) within the Ambient Manufacturing Group (SYMPA) of University of Lorraine, France. His research and teaching is in business process management, process aware information systems, discrete event systems, systems integration and interoperability.

Received: October 29, 2013; Accepted: April 19, 2014.