# Supervisory Control Service - A control approach supporting flexible processes

Eduardo A. P. Santos[1], Rosemary Francisco[1], Maja Pesic[2], Wil van der Aalst[2]

[1]Pontifícia Universidade Católica do Paraná,
Curitiba, Brazil

[2]Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands

April 20, 2010

**Abstract**

In order to be competitive, companies are demanding more flexibility from their Process Aware Information Systems (PAISs). However, increasing flexibility and complexity of modern PAISs usually leads to less guidance for its users and consequently requires more experienced users. For instance, a flexible PAIS allows users to freely choose a specific execution sequence. However, there are no guarantees that the chosen sequence is the best one or at least it conforms to established business rules. In this paper we propose a supervisory control service architecture, which can be used to support end users of flexible PAISs during process execution by giving a list of disabled (or enabled) events (activities) i.e. at any point in time a list of possible next steps is given. To evaluate the correctness of the supervisory control service the approach has been implemented and tested.
**Key words**: supervisory control, business process, process aware information system, monitoring, operational decision making.

## 1 Introduction

Nowadays there is a consensus that the economic success of an enterprise depends on its ability to react to changes in its environment in a quick and flexible way (Weber et al. 2008). The increase in global competition is pushing enterprises to reduce their response times when launching new products and at the same time offer competitive prices. Diversity, fluctuations in demand, the short life cycle of products due to the frequent introduction of new needs, in addition to the increase in the client's expectations in terms of quality and delivery time, are nowadays the main challenges with which companies have to deal in order to remain competitive and stay in business. For these reasons companies have recognized business agility as a competitive advantage, which is fundamental for being able to cope with the problems today's organizations are facing.

According to Weber et al. (2007) and Weber et al. (2008), Process-aware Information Systems offer promising perspectives in this respect, and a growing interest in aligning information systems in a process-oriented way can be observed. A *Process-Aware Information System* (PAIS) is a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models. Examples of PAISs are workflow

management systems, case-handling systems, enterprise information systems, etc (Dumas et al. 2005). In this context, companies are demanding more flexibility from their PAISs. Whilst the concept of flexibility is relatively simple, its implementation is more difficult to achieve in practice (Schonenberg et al. 2008b; Schonenberg et al. 2008a). PAIS can only capture an abstraction of the business process that they facilitate, and recognition that there is a deviation between this process definition and the "real-life" process that they are intended to support requires external (typically human) input. In general, in flexible PAIS it occurs frequently that users working on a case have the option to decide between several activities that are enabled for that case. However, for all flexibility approaches, *the user support provided by the PAIS decreases with increasing flexibility, since more options are available, requiring users to have in-depth knowledge about the processes they are working on.* Traditionally, this problem is solved by educating users (e.g., by making them more aware of the context in which a case is executed), or by restricting the PAIS by introducing more and more constraints on the order of activities and thus sacrificing flexibility. Both options, however, are not satisfactory and limit the practical application of flexible PAIS (Schonenberg et al. 2008b; Schonenberg et al. 2008a; Schonenberg et al. 2008).

In the last years many works have pointed out the flexibility a key for the successful application of workflow technology. The need for flexible PAISs has been recognized and several competing paradigms have been proposed (Schonenberg et al. 2008) (Pesic, Schonenberg, Sidorova, and van der Aalst 2007). However, many researchers have argued that in many applications is desirable to control the process and to avoid incorrect or undesirable executions of the process. In fact, according to van der Aalst et al. (2009), this matter can be seen as paradox, as users expect flexibly and to feel unconstrained in their actions but also expect support at the same time. Van der Aalst et al. (2008) look at other related problem: within a single process/organisation different degrees of flexibility may be required. For instance, the front-office part of the process may require more flexibility while the back-office part requires more control. Thus, many approaches describe the trade-off between flexibility requiring user assistance. Approaches like recommendation service (Schonenberg et al. 2008a) and prediction service (Dongen et al. 2008) give advices to users based on performance considerations. In fact, these services do not force users to take a particular action, they inform users what is the best execution sequence of activities to be perform.

According to van der Aalst et al. (2007) decision making in PAISs involves build-time and run-time decisions. At build-time, idealized process models are designed based on the organization's objectives, infrastructure, context, constraints, etc. At run-time, this idealized view is missed. For instance, process models generally assume that planned tasks happen within a certain period. When such assumptions are not fulfilled, users must make decisions regarding alternative arrangements to achieve the goal of completing the process within its expected time frame or to minimize tardiness. So, as flexibility increases in PAISs, users get more freedom to select suitable tasks for execution. Users can choose, according to their own decision, a specific execution sequence. However, they get no guarantees that the chosen sequence is the best one or at least it conforms to established business rules. Also, on top of PAIS there is no control or only has control in the form of very local rules. In this case, a specific execution could not immediately cause a violation, but the violation could become inevitable. In this sense, we propose to add more global business rules in order to

support users during execution of activities. We do not want neither to remove flexibility on PAISs or use classical restrictive workflow systems, we want to give advices to users in such way that business rules will not be violated.

We propose the *Supervisory Control Service* (SCS) coupled to PAIS. This service has to provide additional business rules according to an observed log event. These business rules aim to inform the users what has to be done (or what should not be done) when processing a case (e.g., choose among enabled activities which one must be executed). The SCS provides information in the form of disabled activities, saying to the user which activities cannot be executed because they violate some control rule. In our paradigm, the SCS can tell the users what cannot happen but it is not be able to force or to choose which event must be executed, i.e. it can block but not force users. For example, after an observed event, three events (activities) can happen, but according to some control rule, the SCS disables one of them. The users has to choose one of the two other activities to continue processing a case. The SCS uses an idea similar to the recommendation service proposed in Schonenberg et al. (2008). The SCS provides information to a user about which activities he/she should avoid in order to achieve a certain group of specifications. Thus, at run time, cases are created and executed considering the constraints imposed by the process model. The SCS imposes additional constrains to such cases according to specifications that were not established in the process model. These specifications can be changed when a redefinition of control rules is necessary. As a recommendation service, our approach is useful when in the process model multiple activities are enabled during execution of a case. Thus, the users can receive an 'on-line' guide from the SCS that tells them which activities should be avoided.

The remainder of this paper is structured as follows. In Section 2, we present an overview of supervisory control service. In Section 3 we present the supervisory control theory and the control architecture of SCS. Then, in Section 4 we show the architecture of SCS and its formal modeling. In Section 5 an example is presented. The implementation of SCS as a Plug-in for Process Mining Framework is described in Section 6. Finally, in sections 7 and 8 we discuss related work and provide conclusions, respectively.

## 2 Overview of supervisory control service

Figure 1 illustrates the Supervisory Control Service (SCS) supporting users of PAISs. In general, each business process to be supported is described as a process model in the respective PAIS. The SCS can be used when the process model provides users a certain freedom to manoeuvre, even considering the constraints imposed by the process model. For example, when multiple activities are enabled during execution of a case. In this situation, users can decide in which order they will execute enabled tasks. It always will be expected that users decide to a better execution sequence, in order to reach the the specific goals. However, considering the lack of information about the process that they are performing, users can take other decisions. Unfortunately, some of these decisions can not be satisfactory in the sense that they can violate some established rules or decrease the performance of the related process. The basic idea of the SCS is to restrict the set of enabled activities during process execution, limiting the level of manoeuvre to the users. Control

rules or specifications are defined in order to establish the behavior of the SCS. Thus, the SCS guides users by disabling activities that would violate predefined goals or business rules. It can be seen as a closed loop system (as in control theory), as the SCS plays the role of controller and the PAIS plays the role of the system to be controlled.

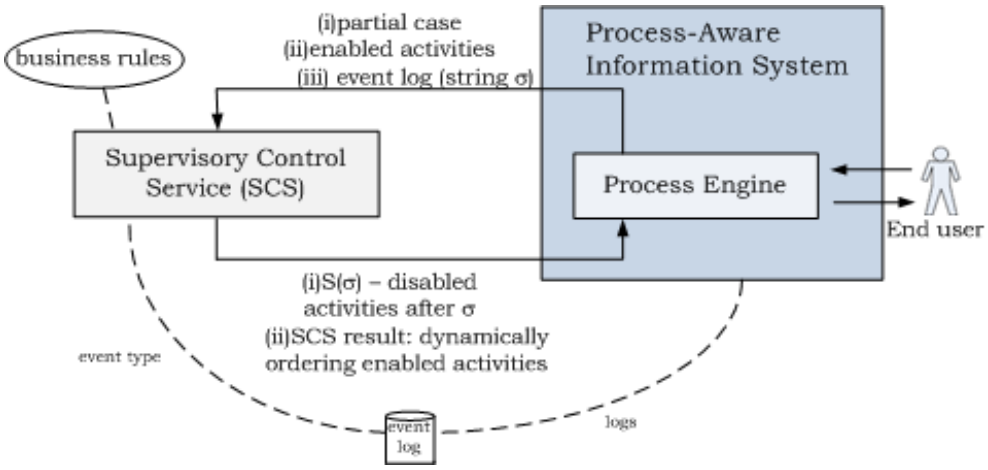


Figure 1: Overview of supervisory control service

In order to implement the SCS, some requisites are necessary. Firstly, we consider any PAIS can generate events log. At run-time, PAIS records information about executed activities in event logs. Typically, event logs contain information about start and completion of activities, their ordering, resources which executed them and the cases (i.e. process instances) they belong to (Van der Aalst et al. 2003). In our approach we consider $\Sigma$ to represent the set of event types associated to a log. The set $\Sigma$ can be seen as the "alphabet" of a language and event sequences can be thought of as "words" in that language. In this framework, we can pose questions such as "Can we control a PAIS in such way that it speaks a given language?" or "Which language does this PAIS should avoid?". Different from conformance analysis (Rozinat and van der Aalst 2008), where it is assumed that a task is associated to at most one type of log event (typically a complete event), in our approach we consider a task logs events at a more fine-grained level. For example, the start, the completion, the suspend of an activity are typical events considered to SCS goals. Secondly, we consider SCS working on a real time mode. Thus, the PAIS should be equipped with a logging mechanism and the SCS must be connected to PAIS. We consider that any information system can offer this information in some form. Note that this is a reasonable assumption as any PAIS has a work list handler (to offer work) and some event logging facility. In addition, a communication infrastructure is necessary to support online connection between the PAIS and the SCS. Thirdly, a well-consolidated formalism to build the engine of SCS is fundamental. As PAISs become more and more complex, in many situations one must deal with a large set of (complex) business rules. The consequence is increasing of complexity the SCS to be constructed. Thus, the formal foundation used to build the SCS must deal with issues as blocking, optimality (what is the best solution?), implementation feasibility, and so on.

As illustrated in Figure 1, the SCS works as following. The process engine generates a set of information elements which is completely observed and processed by the SCS. This set is formed by

a string $\tau$ (an event sequence), a list of enabled activities (after $\tau$) and a case identifier. According to control rules established in SCS, a list of disabled activities is sent to process engine (we call this list the control action at $\tau$). Thus, SCS provides a "map" or "guide" to users, informing them which activities can be executed and which activities cannot. Considering that SCS always allows to reach a target state (or complete every established task), the users can chose one activity from the list of remained enabled activities. In real life flexible processes, with increasing complexity there are so many options for users, that user support becomes fundamental. Restricting these options based on control rules can ensure correctness and support efficiency. At the same time, as the SCS engine can easily and systematically be modified (if control rules or specifications change), this approach is declarative making it very flexible and customizable. For example, if some control rule must be modified, included or excluded, it is easy to synthesize a new controller. This task can be performed automatically using our approach described in the next section.

# 3 Supervisory control theory

The framework described in previous section gives the basis for the design of Supervisory Control Services (SCSs). In this section we introduce the *Supervisory Control Theory* (SCT) proposed by Ramadge and Wonham (1989) as a formalism for supervisor's synthesis. Supervisory control theory has been developed in recent decades as an expressive framework for the synthesis of control for Discrete-Event Systems (DES). In SCT, the open-loop (term familiar in control theory) behavior of a DES, called *plant*, is modeled by an *automaton*. The restrictions to be imposed on the plant can be expressed in terms of a language representing the admissible behavior and it is named specification (Ramadge and Wonham 1987). The Ramadge-Wonham model (or RW model) provides computational algorithms for the synthesis of a minimally restrictive supervisor that constrains the behavior of the plant 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 reach a marked state. While the admissible language can be viewed as a safety specification (assuring that nothing "bad" happens), nonblocking can be interpreted as a liveness specification that ensures that the supervisor will not prevent the completion of a task (something "good" happens) (Queiroz et al. 2005). Note that in Ramadge-Wonham model supervisors cannot force the completion of tasks. This corresponds to reality where we can not force the occurrence of events.

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. Briefly, in RW model a Discrete Event System (DES) is modeled as the generator of a formal language, the control feature being that certain events (transitions) can be disabled by an external controller. The idea is to construct this controller so that the events it currently disables depend in a suitable way on the past behavior of the generating DES. In this way the DES 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 the 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, our SCS uses the RW model as the basic

formalism to build its control engine. Note that in our approach the *process engine is the plant* in RW model (system to be controlled) and *the SCS is considered the controller.*

## 3.1 Formal languages and automata

This section gives a brief introduction to theory used in RW model. This description is classical and similar introductions can be found in Carroll and Long (1989), Cassandras and Lafortune (2008), and Hopcroft et al. (2006). We begin by viewing the event set $\Sigma$ of a DES as an alphabet. We will assume that $\Sigma$ is finite. A sequence of events taken out of this alphabet forms a *word* or *string*. A string consisting of no events is called the empty string and is denoted by $\epsilon$. The length of a string is the number of events contained in it, counting multiple occurrences of the same event. If $s$ is a string, we will denote its length by $|s|$. By convention, the length of the empty string $\epsilon$ is taken to be zero, i.e. $|\epsilon| = 0$.

A language defined over an event set $\Sigma$ is a (possibly infinite) set of finite-length strings formed from events in $\Sigma$. The key operation involved in building strings, and thus languages, from a set of events $\Sigma$ is concatenation. The concatenation $uv$ of two strings $u$ and $v$ is the new string consisting of the events in $u$ immediately followed by the events in $v$. The empty string $\epsilon$ is the identity element of concatenation, i.e., $u\epsilon = \epsilon u = u$ for any string $u$. Let us denote by $\Sigma^*$ the set of all finite strings of elements of $\Sigma$, including the empty string $\epsilon$; the $*$ operation is called the Kleene-closure. Observe that the set $\Sigma^*$ is countably infinite since it contains strings of arbitrarily long length. A language over an event set $\Sigma$ is therefore a subset of $\Sigma^*$. In particular, $\emptyset$, $\Sigma$, and $\Sigma^*$ are languages. We conclude this discussion with some terminology about strings. If $tuv = s$ with $t$, $u$, $v \in \Sigma^*$, then: $t$ is called a prefix of $s$, $u$ is called a substring of $s$, and $v$ is called a suffix of $s$.

The usual set operations, such as union, intersection, difference, and complement with respect to $\Sigma^*$, are applicable to languages since languages are sets. In addition, we will also use the following operations: Concatenation: Let $L_a$, $L_b \subseteq \Sigma^*$, then $L_aL_b := \{s = s_as_b \in \Sigma^* | s_a \in L_a \wedge s_b \in L_b\}$. In words, a string is in $L_aL_b$ if it can be written as the concatenation of a string in $L_a$ with a string in $L_b$. Prefix-closure: Let $L \subseteq \Sigma^*$, then $\overline{L} := \{s \in \Sigma^* | (\exists t \in \Sigma^*) st \in L\}$. In words, the prefix closure of $L$ is the language denoted by $\overline{L}$ and consisting of all the prefixes of all the strings in $L$. Note that $L \subseteq \overline{L}$. $L$ is said to be prefix-closed if $L = \overline{L}$. Thus language $L$ is prefix-closed if any prefix of any string in $L$ is also an element of $L$. Kleene-closure: Let $L \subseteq \Sigma^*$, then $L^* := \{\epsilon\} \cup L \cup LL \cup LLL \ldots$ This is the same operation that we defined above for the set $\Sigma$, except that now it is applied to set $L$ whose elements may be strings of length greater than one. An element of $L^*$ is formed by the concatenation of a finite (but possibly arbitrarily large) number of elements of $L$; this includes the concatenation of "zero" elements, that is, the empty string $\epsilon$. Note that the $*$ operation is idempotent: $(L^*)^* = L^*$.

A language is a formal way of describing the behavior of a DES. It specifies all admissible sequences of events that the DES is capable of "processing" or "generating". In this sense the representation of a DES using languages can be thought as a external behavior model, in which the DES's trajectory is described. However, the use of languages is computational limited. The difficulty here is that "simple" representations of languages are not always easy to specify or work

with. To deal with this issue *automata theory* is used.

An automaton is a device that is capable of representing a language according to well defined rules. The simplest way to present the notion of automaton is to consider its directed graph representation, or state transition diagram. A Deterministic Automaton, denoted by G, is a six-tuple $G = \{Q, \Sigma, \delta, \Gamma, q_o, Q_m\}$, where: Q is the set of states; $\Sigma$ is the finite set of events associated with G; $\delta : Q \times \Sigma \rightarrow Q$ is the transition function: $\delta(q, e) = q'$ means that there is a transition labeled by event e from state q to state q'; in general, $\delta$ is a partial function on its domain; $\Gamma : Q \rightarrow 2^{\Sigma}$ is the active event function (or feasible event function); $\Gamma(q)$ is the set of all events e for which $\delta(q, e)$ is defined and it is called the active event set (or feasible event set) of G at q; $q_0$ is the initial state; $Q_m \subseteq Q$ is the set of marked states.

The connection between languages and automata is easily made by inspecting the state transition diagram of an automaton. Consider all the directed paths that can be followed in the state transition diagram, starting at the initial state; consider among these all the paths that end in a marked state. This leads to the notions of the languages generated and marked by an automaton. The language generated by $G = \{Q, \Sigma, \delta, \Gamma, q_0, Q_m\}$ is $L(G) = \{s \in \Sigma^* | \delta(q_0, s)\}$ is defined. The language marked by $G$ is $L_m(G) := \{s \in L(G) | \delta(q_0, s) \in Q_m\}$. The language $L(G)$ represents all the directed paths that can be followed along the state transition diagram, starting at the initial state; the string corresponding to a path is the concatenation of the event labels of the transitions composing the path. Therefore, a string $s$ is in $L(G)$ if and only if it corresponds to an admissible path in the state transition diagram, equivalently, if and only if $\delta$ is defined at $(q_0, s)$. If $\delta$ is a total function over its domain, then necessarily $L(G) = \Sigma^*$. We will use the terminology active event to denote any event in $\Sigma$ that appears in some string in $L(G)$. Note that not all events in $\Sigma$ need be active. The second language represented by $G$, $L_m(G)$, is the subset of $L(G)$ consisting only of the strings s for which $\delta(q_0, s) \in Q_m$, that is, these strings correspond to paths that end at a marked state in the state transition diagram. The language marked is also called the language recognized by the automaton, and we often say that the given automaton is a recognizer of the given language. Figure 2 shows an automaton and its generated and marked languages.

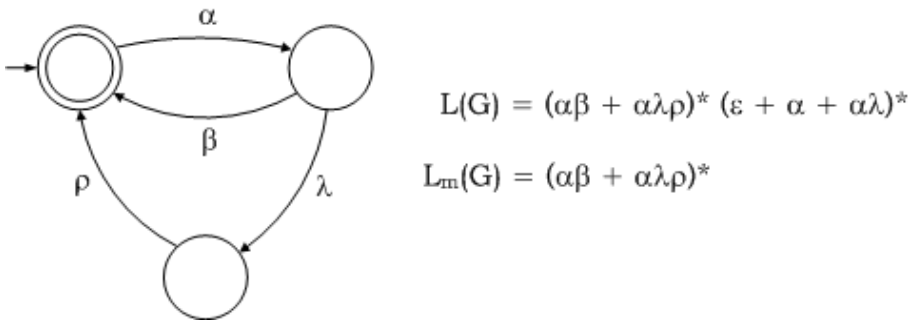


Figure 2: Example of generated and marked language of an automaton

There are two approaches to deal with DES representation using automata: a global approach and a local approach. In the global approach a DES is considered as a whole, and the set of all event sequences build a unique model. In large scale systems this approach can become complex. In addition, whenever a DES must be modified (for example, by including or excluding a specific

subsystem), the model must be modified as a whole. In the local approach a DES is supposed to be constituted of subsystems and each one of them can be represented by a local automaton. To obtain a global representation of such DES it is necessary to compose the local models. A local approach aims to facilitate representing large scale DESs, considering that it is easier to obtain small models. Also, whenever a DES must be modified it is only necessary rebuild a specific local model(s). A key operation to apply the local approach is *the composition* of automata. In general, when modeling systems composed of interacting components, the event set of each component includes private events that pertain to its own internal behavior and common events that are shared with other automata and capture the coupling among the respective system components (Cassandras and Lafortune 2008). The standard way of building models of entire systems from models of individual system components is by parallel composition.

Consider the two automata $G_1 = \{Q_1, \Sigma_1, \delta_1, \Gamma_1, q_{01}, Q_{m1}\}$ and $G_2 = \{Q_2, \Sigma_2, \delta_2, \Gamma_2, q_{02}, Q_{m2}\}$. The parallel composition (or synchronous product) of $G_1$ and $G_2$ is the automaton $G_1||G_2 := Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, \Gamma_{1||2}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$ where $Ac$ stands for taking the accessible part of $G_1||G_2$:

$$\delta\left((q_1, q_2), e\right) := \begin{cases} \left(\delta_1\left(q_1, e\right), \delta_2\left(q_2, e\right)\right) & \text{if } e \in \Sigma_1 \cap \Sigma_2 \wedge e \in \Gamma_1\left(q_1\right) \cap \Gamma_2\left(q_2\right) \\ \left(\delta_1\left(q_1, e\right), q_2\right) & \text{if } e \in \Sigma_1 \wedge e \notin \Sigma_2 \wedge e \in \Gamma_1\left(q_1\right) \\ \left(q_1, \delta_2\left(q_2, e\right)\right) & \text{if } e \in \Sigma_2 \wedge e \notin \Sigma_1 \wedge e \in \Gamma_2\left(q_2\right) \\ \text{undefined}, & \text{otherwise} \end{cases} \tag{1}$$

and thus $\Gamma_{1||2}(q_1, q_2) = [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \backslash \Sigma_2] \cup [\Gamma_2(q_2) \backslash \Sigma_1]$.

In the parallel composition, a common event, that is, an event in $\Sigma_1 \cap \Sigma_2$, can only be executed if the two automata both execute it simultaneously. Thus, the two automata are synchronized on the common events. The private events, that is, those in $(\Sigma_2 \backslash \Sigma_1) \cup (\Sigma_1 \backslash \Sigma_2)$, are not subject to such a constraint and can be executed whenever possible. In this kind of interconnection, a component can execute its private events without the participation of the other component; however, a common event can only happen if both components can execute it. If $\Sigma_1 = \Sigma_2$, then all transitions are forced to be synchronized. If $\Sigma_1 \cap \Sigma_2 = \emptyset$, then there are no synchronized transitions and $G_1||G_2$ is the concurrent behavior of $G_1$ and $G_2$. This is often termed the shuffle of $G_1$ and $G_2$. Figure 3 shows an example of synchronous product of two automata.

## 3.2 Supervisory control

The situation under consideration in this section is that of a given DES, modeled at the untimed (or logical) level of abstraction, and whose behavior must be modified by feedback control in order to achieve a given set of specifications. Let us assume that the given DES 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 DES. 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 alter the behavior of $G$ we introduce a supervisor; supervisors will be denoted by $S$. Note that we separate the plant (term usually used in industrial automation) $G$

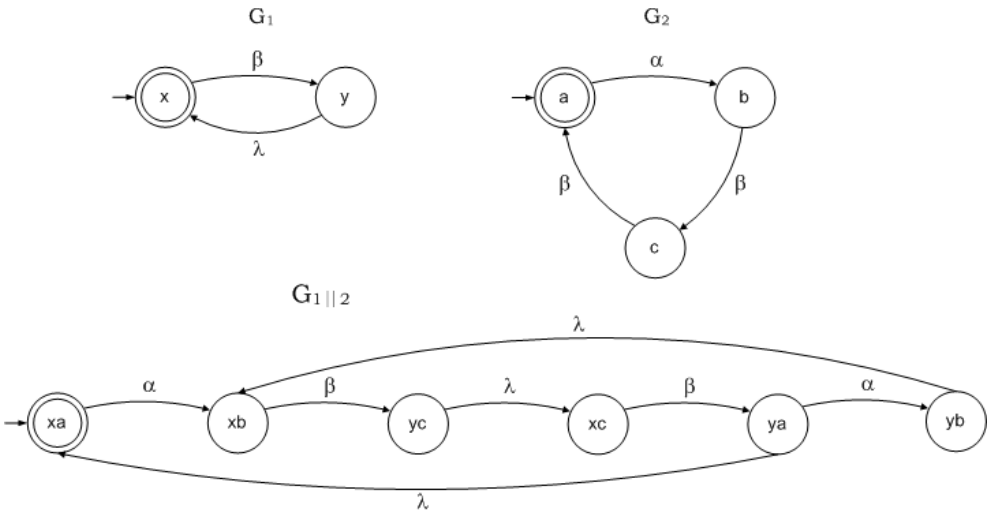


Figure 3: Example of synchronous product of two automata

from the controller (or supervisor) $S$, as is customary in control theory. This raises two questions: (a) What is the meaning of specifications? and (b) How does $S$ modify the behavior of $G$?

The language $L(G)$ contains strings that may be unacceptable because they violate some rule or nonblocking condition that we wish to impose on the system. It could be that certain states of $G$ are undesirable and should be avoided. These could be states where $G$ blocks, via a deadlock or a 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. These substrings may violate a desired ordering of certain events, for example, requests for the use of a common resource should be granted in a first-come first-served manner. Thus, we will be considering sublanguages of $L(G)$ that represent the legal or admissible behavior for the controlled system. In some cases, we may be interested in a range of sublanguages of $L(G)$, of the form $L_r \subset L_a \subseteq L(G)$ where the objective is to restrict the behavior of the system to the range delimited by $L_r$ and $L_a$; here, $L_a$ is interpreted as the maximal admissible behavior and $L_r$ as the minimal required behavior. In both the inclusion and range problems, we could have the additional requirement that blocking does not occur.

Ramadge and Wonham 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.

## 3.3 Controlled DES

In Ramadge and Wonham model, 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* $\Sigma_{uc}$. An event is classified

as controllable if its occurrence can be disabled by supervisor. It is classified as uncontrollable in the opposite case. According to (Cassandras and Lafortune 2008), an event might be modeled as uncontrollable because it is inherently unpreventable (for example, a fault event); or it models a change of sensor readings not due to a command; it cannot be prevented due to hardware or actuation limitations; or it is modeled as uncontrollable by choice, as for example when the event has high priority and thus should not be disabled or when the event represents the tick of a clock. Formally, a supervisor $S : L(G) \rightarrow 2^{\Sigma}$ is a function that maps from the sequence of generated events to a subset of controllable events to be disabled. The behavior of the plant $G$ under supervision of supervisor $S$ is represented by the language marked by the automaton $S/G$ (synchronous product of $S$ and $G$). The necessary and sufficient conditions for the existence of a supervisor are presented in Ramadge and Wonham (1989).

The key existence result for supervisors in the presence of uncontrollable events is the Controllability Theorem. Consider a DES $G = \{Q, \Sigma, \delta, \Gamma, q_0, Q_m\}$ where $\Sigma_{uc} \subseteq \Sigma$ is the set of uncontrollable events. Let $K \subseteq L(G)$, where $K \neq \emptyset$. Then there exists supervisor $S$ such that $L_m(S/G) = K$ if and only if $\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}$. This condition on $K$ is called the *controllability condition*. The necessary and sufficient condition for the existence of a non-blocking supervisor $S$ that reaches a given specification $K \subseteq L_m(G)(L_m(S/G) = K)$ is the controllability of $K$. The language expression for the controllability condition can be rewritten as follows: for all $s \in \overline{K}$, for all $\sigma \in \Sigma_{uc}$, $s\sigma \in L(G) \Rightarrow s\sigma \in \overline{K}$. Thus an occurrence of an uncontrollable event, after a string in $K$, keeps this sequence in $K$.

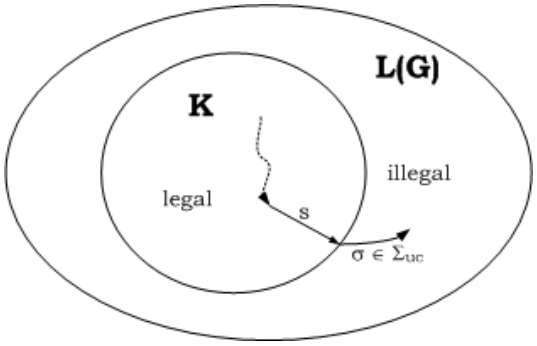


Figure 4: Illustration of Controllability condition

If a given language $K$ is not controllable, it is desirable to find the "largest" sublanguage of $K$ that is controllable, where "largest" is in terms of set inclusion. The question is: Does such a sublanguage of $K$ exist? According to Ramadge and Wonham (1989), the class of controllable sublanguages in $L(G)$ is $C(M, G) = \{K | K \subseteq L_m(G)\}$ and K is controllable with respect to $G$ and it contains a (unique) supremal element named Sup$C(M, G)$.

A possible representation of a supervisor is a pair $S = (S, \Phi)$, where $S = (Q^S, \Sigma^S, \delta^S, q_0^S, Q_m^S)$ is an automaton with $\Sigma^S = \Sigma^G = \Sigma$ and $\Phi : Q^S \rightarrow 2^{\Sigma}$ is an output map that specifies the subset of controllable events to be disabled at each state of the automaton representing the supervisor. Usually the automaton representing the supervisor is the automaton $S/G$ itself. Minhas (2002) and Su and Wonham (2004) deal with the reduction of supervisors. The reduction of the supervisor $S$ is the achievement of another representation of it, namely $S = (S_r, \Phi_r)$, where the automaton

$S_r$ has a smaller number of states than the automaton $S/G$ and such that this reduction does not affect the control action of the supervisor.

### 3.4 Supervisory control architecture

Figure 5 presents a Supervisory Control Architecture. The SCA as in Ramadge and Wonham (1989) assumes that the plant (taking place of the system to be controlled) spontaneously generates all events and the role of the supervisors is to enable/disable controllable events. Assume that all the events in $\Sigma$ executed by $G$ are observed by supervisor $S$. Thus, in Figure 5, $\sigma$ is the string of all events executed so far by $G$ and $\sigma$ is entirely seen by $S$. The control paradigm is as follows. The transition function of $G$ can be controlled by $S$ in the sense that the controllable events of $G$ can be dynamically enabled or disabled by $S$.

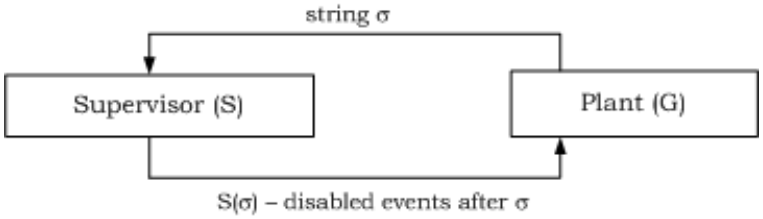


Figure 5: Supervisory control architecture

For each $\sigma \in L(G)$ generated so far by $G$ (under the control of $S$), $S(\sigma) \cap \Gamma(\delta(q_0, \sigma))$ is the set of enabled events that $G$ can execute at its current state $\delta(q_0, \sigma)$. In other words, $G$ cannot execute an event that is in its current active event set, $\Gamma(\delta(q_0, \sigma))$, if that event is not also contained in $S(\sigma)$. In view of the partition of $\Sigma$ into controllable and uncontrollable events, we will say that supervisor $S$ is admissible if for all $\sigma \in L(G)$:

$$\Sigma_{uc} \cap \Gamma(\delta(qx_0, \sigma)) \subseteq S(\sigma) \tag{2}$$

which means that $S$ *is not allowed to ever disable a feasible uncontrollable event*. From now on, it will be only consider admissible supervisors. $S(\sigma)$ is called the control action at $\sigma$. $S$ is the control policy. Given $G$ and admissible $S$, the resulting closed-loop system is denoted by $S/G$ (read as $S$ controlling $G$). The controlled system $S/G$ has generated and marked languages. These two languages are simply the subsets of $L(G)$ and $L_m(G)$ containing the strings that remain feasible in the presence of $S$. The language generated by $S/G$ is defined recursively as follows: 1. $\epsilon \in L(S/G)$; 2. $[(s \in L(S/G)) \wedge (s\sigma \in L(G)) \wedge (\sigma \in S(s))] \Leftrightarrow [s\sigma \in L(S/G)]$. The language marked by $S/G$ is defined as follows: $L_m(S/G) := L(S/G) \cap L_m(G)$.

Remark that the supervisory control architecture shown in Figure 5 is a basis for defining the SCS as illustrated in Figure 1. In fact, the supervisor obtained using SCT approach will be the main entity implemented in supervisory control service. However, in order to implement the control system under the SCT approach it is necessary to deal with some limitations of Ramadge and Wonham model. The complexity of supervisors' synthesis, although polynomial in the number of states of the plant and specification models, is an obstacle in applications since the number

of states that represent the system increases exponentially with the number of its components. This restricting factor, that is specially relevant for large scale systems, has been considered by several authors that attempt to overcome these computational difficulties by exploiting different aspects of the system, as Local Modular Control (LMC) (De Queiroz and Cury 2000b), hierarchical control (Wong and Wonham 1996) (Zhong and Wonham 1990), modular control (Ramadge and Wonham 1988) (**?**), symmetry (Cury and Eyzell 2001). In addition, despite the fact that the Ramadge and Wonham model is a well established theory, the gap between the theoretical model and implementation methodologies still remains. Thus, to deal with supervisors' synthesis we use LMC approach (De Queiroz and Cury 2000b), which is an extension of Ramadge and Wonham model that consider a distributed control architecture. Also, as a contribution of this work, we propose a supervisory control architecture in order to implement the results of LMC.

## 3.5 Modular supervisory control

According to the LMC approach (De Queiroz and Cury 2000b), the system to be controlled is modeled by a Product System Representation (PSR), i.e., by a set of asynchronous subsystems $G_i|i \in I$ such that all pairs of subsystems in this set have disjoint alphabets (Ramadge and Wonham 1989). The behavior of each subsystem is represented by an automaton $G_i = (\Sigma^{Gi}, Q^{Gi}, \delta^{Gi}, q_0^{Gi}, Q_m^{Gi})$, such that the behavior of the entire system to be controlled is obtained by the synchronous product of all subsystems of the PSR, i.e. $G = ||_{\forall i \in I} G_i$. The whole set of events is $\Sigma = \cup_{i \in I} \Sigma^{Gi}$. Considering a subsystem in $G_i|i \in I$, $\Sigma c^{Gi}$ denotes its set of controllable events and $\Sigma uc^{Gi}$ its set of uncontrollable events. Furthermore, large scale systems are characterized by having several specifications, each acting on just part of the global system. Generally, these specifications attempt to synchronize some concurrent subsystems. This peculiarity of PSR is exploited by local modular control in the following.

The LMC approach states that, instead of synthesizing a single global supervisor that satisfies the entire set of specifications, one local supervisor is synthesized in order to satisfy each specification. Each one of the local supervisors restricts the behavior of a part of the system to be controlled. This part is the local plant corresponding to the considered supervisor. A local plant $Gl_j$ is obtained by performing the synchronous product of the subsystems (activities) which share events with the considered specification. The synthesis of a local supervisor $S_j$ is performed considering the corresponding specification $E_j$ ($\Sigma_j \subseteq \Sigma$) and its local plant $Gl_j$ ($G_j = \big\|_{i \in N_j} G_i$), with $N_j = \{k \in N \,|\Sigma_k \cap \Sigma_j \neq \emptyset\}$. Thus, the local plant $Gl_j$ joins only the subsystems of the original composed system that are directly restricted by $E_j$. By using this procedure, it is possible to synthesize a local supervisor for each one of the established specifications. If at least one local supervisor in the set $S_j|j \in J$ disables the occurrence of an event, then the occurrence of this event is disabled in $G$. Even when all local supervisors are non-blocking, the concurrent control action of the whole set of supervisors may still result in the blocking of the entire system. Therefore, after accomplishing the synthesis procedure, it is necessary to verify the modularity property of the set of supervisors as stated in (De Queiroz and Cury 2000a). When this condition holds, it is necessary and sufficient to assure that the modular approach does not cause any loss of performance with

relation to the monolithic approach.

According to De Queiroz and Cury (2000b), when the condition of modularity does not hold, the local modular approach cannot be applied directly, but it can still be explored by other methodologies. The modular-control-and-coordination approach presented in (Wong and Wonham 1998), modular control with priorities (Chen et al. 1995), and the scheme for conflict resolution presented in (Wong et al. 1995) are examples of such methodologies. On the other hand, another great advantage of representing the specifications in terms of the local system is that, for some special cases, the designer can identify modularity without the obligation of additional calculations. An evident example is when the local specifications (composed with the affected subsystems) have disjoint alphabets. Also note that, although in the original problem no restrictions are imposed on the information structure for control, the results presented point out that the locally modular synthesis for composed systems induces a natural decentralized structure for supervisors. In fact, each local supervisor only needs to exchange information with its corresponding local plant.

# 4 Development cycle of SCS

The SCS development occurs cyclically in three stages - modeling, synthesis and implementation - up to the moment in which it complies with the real system's requirements. This development approach allows a continuous review of the results obtained in each step. By doing this, the designer can receive a new requirement (for example, a need for processes reconfiguration or for modifying business rules) and build a new control system which will appropriately comply with this new requirement. Figure 6 shows the three stages of the development cycle.

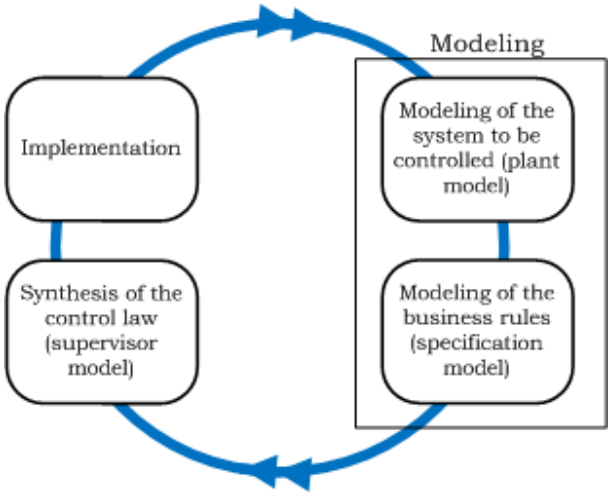


Figure 6: Development cycle of SCS

As presented in previous section, in order to apply the Ramadge and Wonham model it is necessary to identify the system to be controlled and the control rules (specifications) to be imposed. These activities characterizes the first stage of development cycle. At this point it is necessary to recall that the PAIS is the system to be controlled. According to the main concept of SCS, its objective is to restrict the feasible behavior of PAIS users. To do that, the control mechanism implemented in SCS dynamically disables some activities according to the past event sequence

from PAIS. Considering that the control action is related to the tasks, we pose that the system to be controlled can be represented as the set of the activities involved in PAIS. Additionally, as mentioned before, we consider in this abstraction an activity events log at a more fine-grained level. It means that, for control purposes, we are interested in observing intermediate states of an activity, not only its finalization (for some types of analysis such as process mining and conformance checking, usually only the *complete* event is considered). For instance, an activity model might have states with different semantics: 'activity is idle' (waiting for a case), 'activity is being executed' (processing a case), and so on. Note that the control function the supervisory service expects must able to identify the states of activities correctly. It means the states of an activity can be defined in accordance to business rules (in the sense that some events must be observed in order to implement some business rule). For modeling purposes, each activity is associated to an automaton model and then a global model can be obtained through parallel composition.

The second stage consists of applying the synthesis procedure proposed by Ramadge and Wonham (1989) as of models obtained in previous stage. At this point a decision has to be made on which approach will be used: classical (monolithic supervision), modular (Ramadge and Wonham 1988), local modular (De Queiroz and Cury 2000b), decentralized (Rudie and Wonham 1992), hierarchical (Wong and Wonham 1996), and so on. In addition, considering that the objective is to implement the supervisors in information systems, it is advisable and necessary to apply supervisor reduction algorithms as proposed in Vaz and Wonham (1986), Su and Wonham (2004), and Minhas (2002). According to Su and Wonham (2004), this is because the algorithm to compute the maximal controllable language can be much larger than what is actually required for the same control action. The reason for that is the controlled behavior incorporates all the a priori transitional constrains required by control action to enforce the specifications. The problem of finding a simplified proper supervisor equivalent in control action but minimal in terms of size, is evidently of practical interest. This allows a smaller amount of memory and a better control program legibility and at the same time it facilitates modifications and maintenance in the whole control system.

The third stage of development cycle consists of implementing the theoretical results obtained with the application of Ramadge and Wonham model. Basically it is necessary to systematically translate the theoretical supervisor of Ramadge and Wonham model into a computational language. The main issue is to succeed in making the program perform in the same way as the supervisor does in SCT.

According to development cycle of SCS shown in Figure 6 business rules can be defined at design-time and at run-time. At design-time, the process designer can incorporate business rules within a process definition. In this case the SCS and the PAIS are implemented together. At run-time, new requirements or business rules may be necessary to incorporate in SCS. Thus, the control program running in SCS should be modified in order to include new business rules.

## 4.1 Modeling activities

To apply the modular supervisory control approach (De Queiroz and Cury 2000b) at synthesis stage, we consider that a PAIS offers a set of activities. We use the Product System Representation to

model the elements of this set. Each activity is assigned to an automaton representing its behavior. Briefly, each state of this automaton represents a possible intermediate state or a refinement of such activity. Events represent the transition to a state to another one. The initiation and finalization of an activity are examples of events. In general, the modeling of an activity is done considering which information should be observed by SCS. Depending on what business rules should be implemented, a set of intermediate states should be observed. Figure 7 presents three examples of activity models.

The first one, with two states, represents the situation where an activity signalizes the occurrence of event *complete* (it represents a finalization of such activity). The second one is a refinement of the first one, as it includes an intermediate state. In this case, it is necessary observe an activity in one of three states: in initial state it is not being executed, in the second state it is being executed and in the third state it has been finalized. The events *start* and *complete* represent the initiation and the finalization execution of such activity, respectively. The third one includes a fourth state that may represent exceptional situations. For example, this state is reached if the execution of an activity is temporarily halted (through the occurrence of event *suspend*).

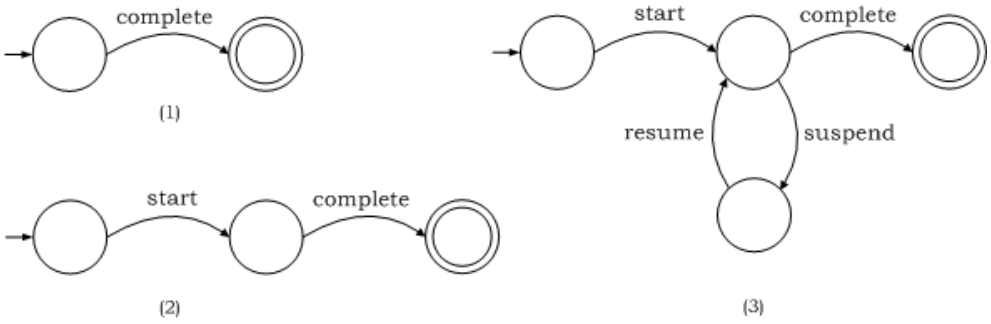


Figure 7: Examples of activity models

The product system representation (PSR) employed in the modeling step preserves the natural modularity and structure of a process description. Notice that the independent behavior of each activity is represented through a corresponding automaton. As a consequence the set of asynchronous automata built reflects the set of activities constituting the process (or part of it) to be controlled. Adding or excluding an activity in a process description only requires adding or excluding the automaton corresponding to this activity in the adopted PSR. Besides, when the implementation of a specific business rule requires to observe a new state of some activity, it is only necessary to include a new state in the corresponding automaton. Despite the fact that the changes previously mentioned will modify the model representing an activity, the impact caused by them can be treated in a simple and objective manner as described.

## 4.2 Modeling business rules

The SCS must ensure that the execution of activities in PAIS is restrained by business rules. Since it is in principle impossible to list all possible business rules, we only present groups of them that occur frequently. Based on the specifications classes proposed by Cassandras and Lafortune (2008), we define four specification classes present in business processes:

1. *Prohibited states*: we identify states in the activities model that cannot occur due to some

restrictions or security. For instance, a state where two activities are simultaneously processing a case must be avoided because a constraint is violated. The business rule model is obtained by simply excluding those states from the activities model. Figure 8 shows an example of such rule. It is considered that the model $G$ is obtained by synchronous product of two automata (each one representing an activity). Each activity is represented by an automaton as shown in Figure 7 (example 2).

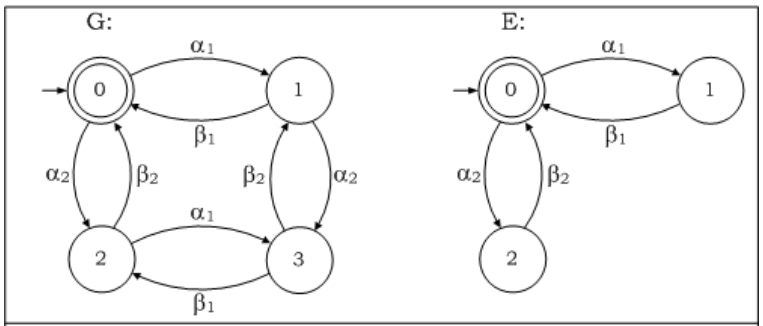


Figure 8: Business rules - prohibited states

2. *Ordering based rules*: express constraints concerning the ordering of events and activities in processes. For instance, an activity must always precede an other activity; an activity must not exceed a limited number of repetitions of a process; a mutual exclusion between two activities. Figure 9 shows some examples of such business rules. It is considered that events labeling the automata (events $\alpha_i$) show in Figure 9 may represent the initiation of activities.
3. *Illegal substring*: if a business rule identifies as illegal all strings of the activities model that contain a specific substring, it is possible to build an automaton that represent this specification. To do that, an algorithm described in Cassandras and Lafortune (2008) can be used. Figure 10 show an example of such business rule.
4. *State Splitting*: If a specification requires remembering how a particular state of the activities model was reached in order to determine what future behavior is admissible, then that state must be split into as many states as necessary. The active event set of each newly introduced state is adjusted according to the respective admissible continuations. Figure 11 show an example of such business rule (Cassandras and Lafortune 2008).

The decomposition of business rules in groups as proposed allows developing libraries of models. This guarantees the minimization of time consumed at the several steps involved in the design of the SCS with similar characteristics. The advantage of this approach, comparing with empirical ones, is to preserve the knowledge in modeling business rules. It also facilitates to transfer such knowledge to other projects and designers. In addition, this would make feasible for business experts to add business rules without the help of specialists in automata theory, but just selecting textual description of a business rule (which is assigned to an automaton).

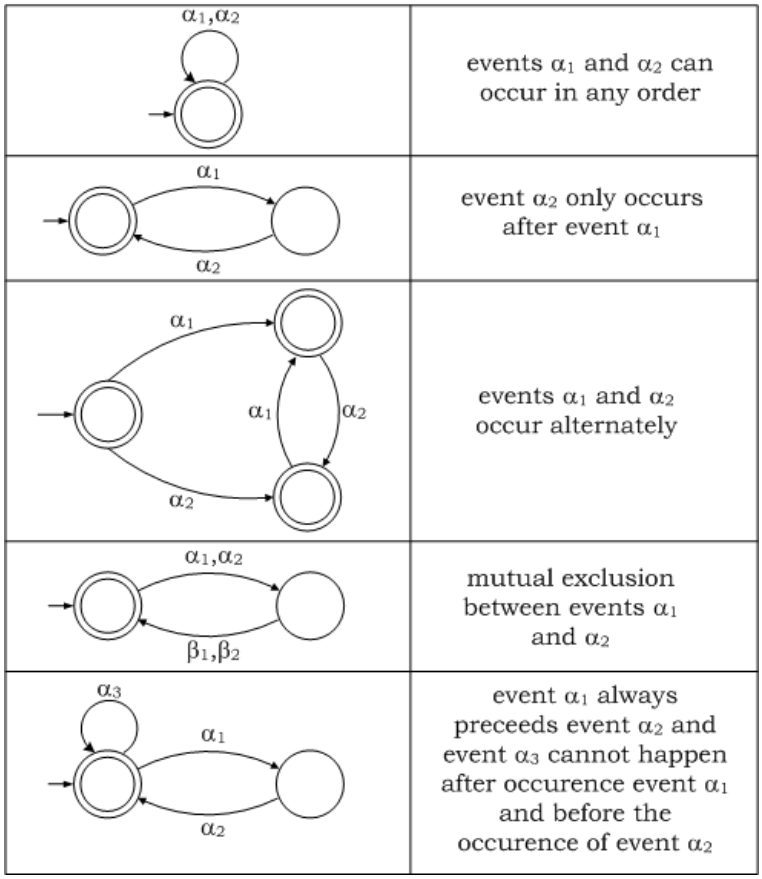


Figure 9: Business rules - ordering based rules

# 5 Example

As an example applying our SCS approach, we use a fictive process, described in Schonenberg et al. (2008). In the example which employees have to do five activities named A, B, C, D and E. The employees can decide in which order to execute these activities. Ideally, the employees finish these as soon as possible. All activities have a fixed duration, however, activities B and C use the same database application and if B is directly followed by C, then the combined duration of the activities is much shorter, since there is no closing time for B and not set-up time for C, moreover C can use the data provided by B, without data re-entry. Activities A and E, during their execution, can unexpectedly fail. In case both activities fail simultaneously, the repair of activity A has priority over the activity E. Also, for security reasons, activities B and D should not be executed at the same time. It is necessary a mutual exclusion between them. The supervisory control service can guide the employees providing them which activities are disabled in order to accomplish the specifications. Remark that disabling an activity means it is not allowed to occur (control action provided by SCS).

For modeling purposes, we consider each task can be modeled as an automaton having two states: (1) an initial state means the activity is not being executed (a case has not come yet); (2) another state means a case is being processing. In the case of activities A and E, it is necessary to represent a third state, named 'fail'. Activities B, C and D operate as follows. Initially there is no case to be processed. With event *start*, the activity is initiated (state 1 is reached). When it

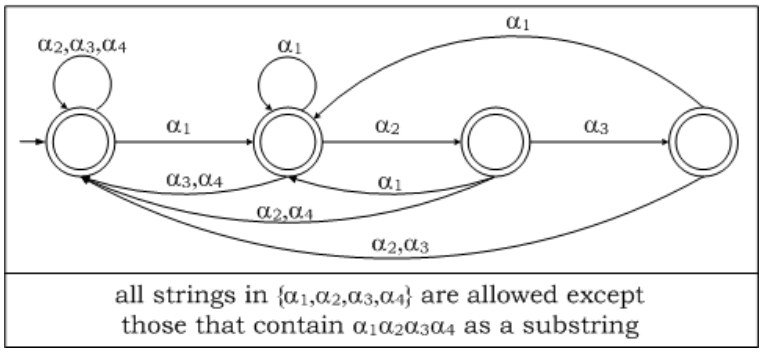


Figure 10: Business rules - illegal substring

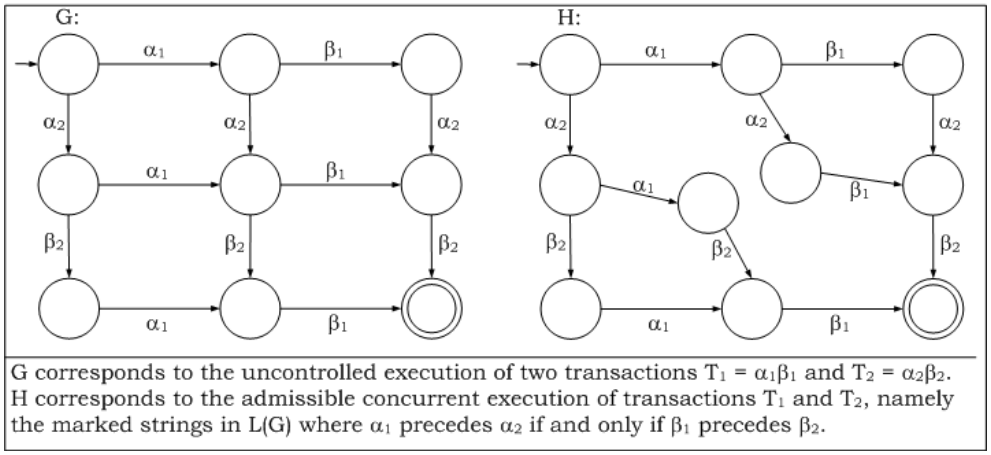


Figure 11: Business rules - state splitting

finishes (signalled by the occurrence of event *complete*) it returns to state 0. Activities A and E operate similarly, but both can suspend and a new state is reached (event *suspend* means that the activity is temporarily halted). After a repair (event *repair*), they return to their corresponding initial state. The automata representing the five activities are named $G_A$, $G_B$, $G_C$, $G_D$ and $G_E$ and are shown in Figure 12. States labeled with 0 represent waiting for a case and states labeled with 1 represent 'executing activity'. In the case of automata $G_A$ and $G_E$, state 2 represents 'activity suspended'. We consider that the events related to beginning of an activity (*start* events) are *controllable* and the events related to ending of an activity (*complete* events) are *uncontrollable*. Thus, when an activity begins, the supervisors cannot disable it until it finishes. Also, the events related to suspending an activity (*suspend* events) are considered uncontrollable (in the sense they are unpredictable) and the events related to repair (*repair* events) are controllable (it is possible to avoid a repair). Also, in automata a controllable event may be indicated by an optional tick on its transition arrow. In automata a controllable event is indicated by a tick on its transition arrow.

The automata shown in Figure 12 model all strings of events that could happen in the open-loop system (i.e. uncontrolled), including undesired ones. We can interpret these automata as representing the semantics of involved activities, without any additional restrictions. However, it is necessary to include some control during the execution of activities in order to meet the specifications. In fact, it can be done avoiding some event sequences. To calculate the control logic that avoids the occurrence of undesired sequences, it is necessary to express the required behaviour

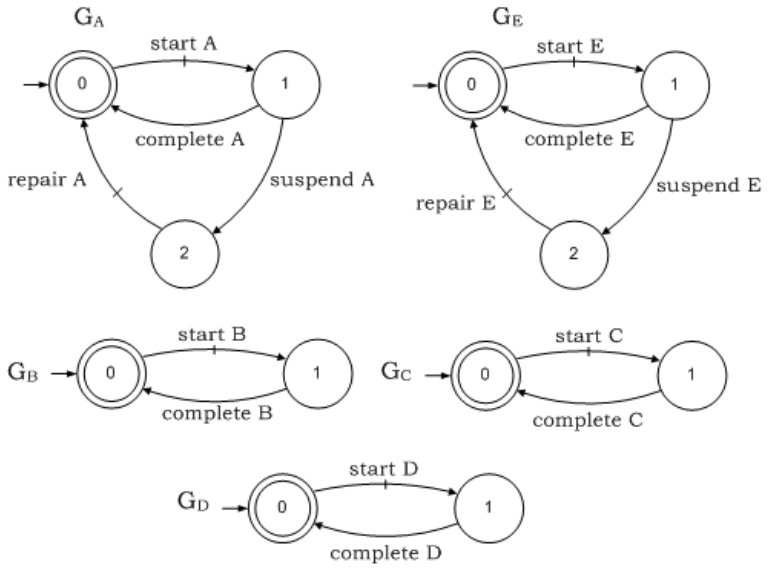


Figure 12: Automata representing activities A,B,C,D and E

in terms of constructs mentioned in Section 4.1. These specifications are built considering the restrictions described previously and can be represented by the automata shown in Figure 13. In order to obtain a shorter execution of the described process, specification $E_1$ establishes that the task B must be performed just after task C has been finished. Notice that in automata $E_1$ event *start C* can only occur after an occurrence of event *complete B*. Specification $E_2$ establishes that the activity A has priority of repair over the activity E in the case of both fail. Automaton $E_2$ shows that repair of activity E (event *repair E*) is only possible after repairing activity A (event *repair A*). Note that event *repair E* is not active in state 1 (in which activity A has failed). Specification $E_3$ establishes a mutual exclusion between activities B and D. In this automaton, after occurrence of event *start B* (*start C*), is not possible an occurrence of event *start C* (*start B*), unless after occurrence of event *complete B* (*complete C*).

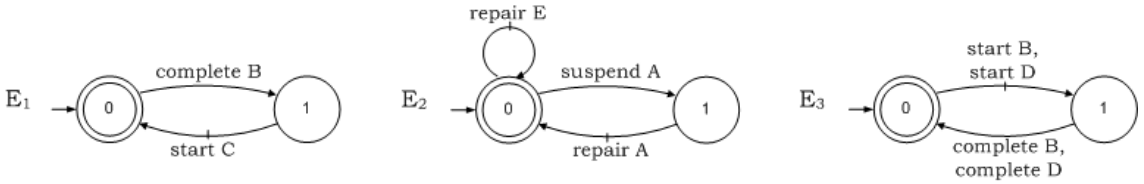


Figure 13: Automata representing specifications

According to Local Modular Control approach, the first step to synthesize local supervisors is to obtain the local plant to each specification. The local plants to $E_1$, $E_2$ and $E_3$ are respectively given by $Gl_1 = G_B||G_C$, $Gl_2 = G_A||G_E$ and $Gl_3 = G_B||G_D$. These local plants are shown in Figure 14. Notice that specifications $E_1$ and $E_3$ are related to the common activity model $G_B$. The states of local plants are labeled with the corresponding states of automata which originated them (through product synchronous operation). For example, the state '00' of local plant $G_1$ corresponds to state 0 both in activities B and C.

Using the algorithms proposed by Ramadge and Wonham (1989), it is possible to obtain three local supervisors, each one guarantees the specification expressed by the corresponding automaton

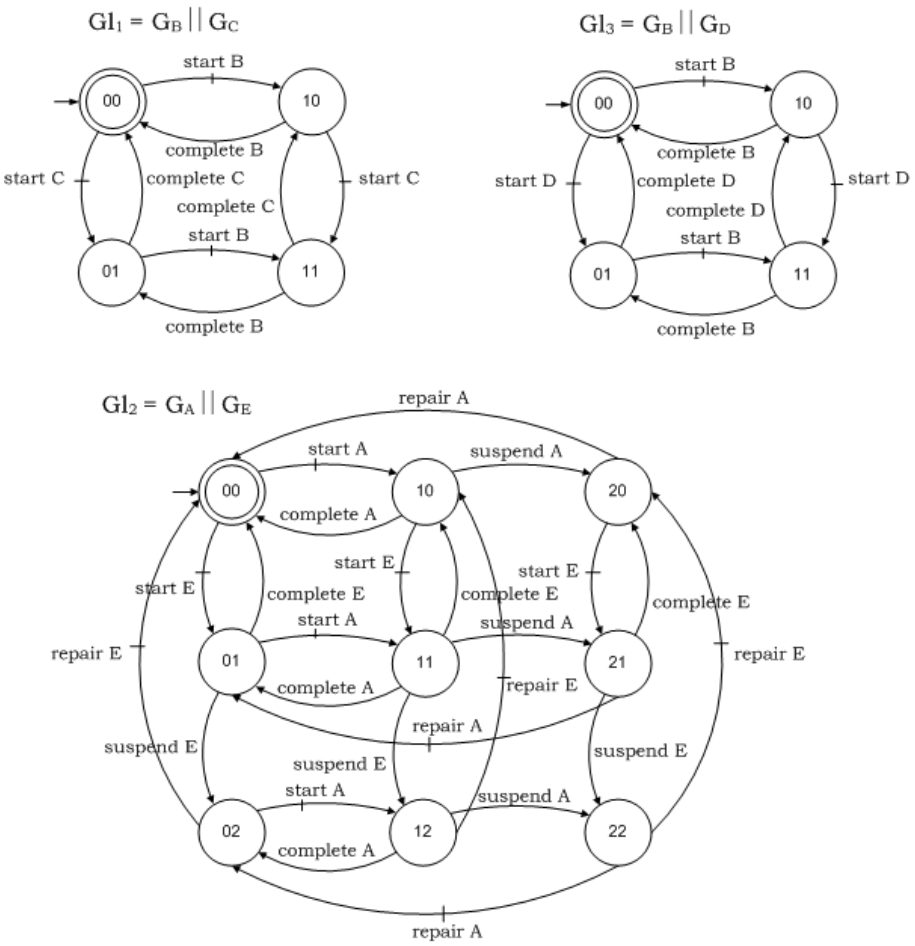


Figure 14: Automata representing local plants

$E_j (j = 1, 2, 3)$. The synthesis of a local supervisor $S_j$ is performed considering the corresponding specification $E_j$ and its local plant $Gl_j$. By using this procedure, it is possible to synthesize a local supervisor for each one of the established specifications. The software TCT (Feng and Wonham 2006) was used to perform the synchronous composition, the synthesis of supervisors and the reducing procedure of supervisors. The obtained supervisors $S_1$, $S_2$ and $S_3$ corresponding to specifications $E_1$, $E_2$ and $E_3$, respectively, are shown in Figure 15. The resulting closed-loop system denoted by $S_i/G_i$ (read as $S_i$ controlling $G_i$) results in the subsets of $L(G_i)$ and $L_m(G_i)$ containing the strings that remain feasible in the presence of $S_i$. For example, only event *start B* is defined for the initial state of supervisor $S_1$ (see Figure 15), whilst events *start B* and *start C* are defined on initial state of local plant $Gl_1$ (see Figure 14). It means $S_1$ controlling $Gl_1$ avoids the occurrence of event *start C* in the initial state of $G_1$ to meet the requirement stated in $E_1$. Remark that according to RW model the automaton $S$ is driven by the occurrence of events in the plant $G$ and the 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 output maps of supervisors shown in Figure 15 are: for $S_1$ is $\Phi_1(0) = start\ C$, $\Phi_1(1) = start\ C$, $\Phi_1(2) = start\ B$, $\Phi_1(5) = start\ B$; for $S_2$ is $\Phi_2(8) = repair\ E$; and for $S_3$ is $\Phi_3(0) = start\ D$, $\Phi_3(1) = start\ D$. For example, in automaton $S_1$ is only defined the occurrence of event *startB* in its initial state and the output map defines event *startC* in the same state. As $S_1$ observes and control the local plant $G_1$ (see Figure 14), it

means that event *startC* is not allowed to occur in the initial state of $G_1$ (only event *startB* is allowed).

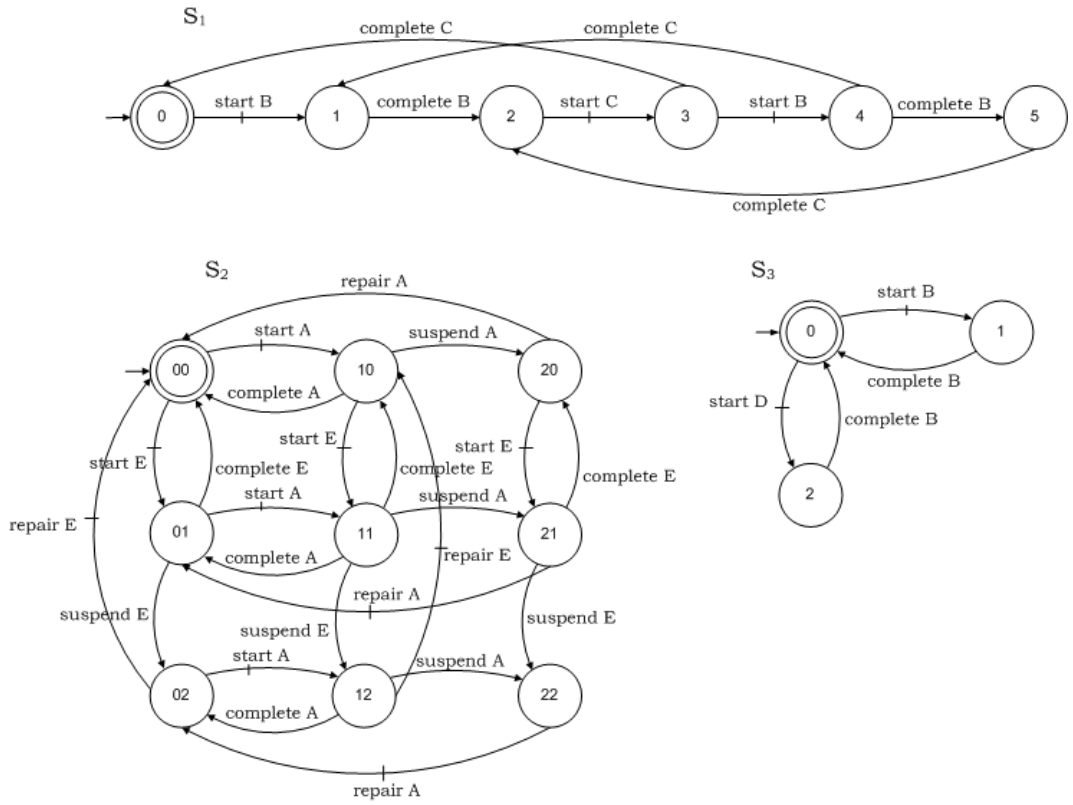


Figure 15: Automata representing local supervisors

## 5.1 Optimizing the control structure

As a final step before the physical implementation, reduction of supervisors is taken into account, for a reduction in the number of states of a supervisor can represent memory economy and clarify the control logic. The reduced supervisor has a smaller number of states and the same control action that corresponding supervisor has. For supervisors $S_1$, $S_2$ and $S_3$ their respective reduced supervisors $Sr_1$, $Sr_2$ and $Sr_3$ are obtained by reduction algorithm proposed by Su and Wonham (2004). Although this algorithm has exponential complexity, it becomes feasible for local modular supervisors that usually have a smaller number of states. Each supervisor may be represented by a corresponding pair $(S_j, \Phi_j)$. Considering the reduced supervisors $Sr_j$ shown in Figure 16, their output maps are: for $Sr_1$ is $\Phi r_1(0) = start\ C$, $\Phi r_1(1) = start\ B$; for $Sr_2$ is $\Phi r_2(1) = repair\ E$; and for $Sr_3$ is $\Phi r_3(1) = start\ B$,*start* $D$. Figure 16 illustrates the set of reduced local modular supervisors, with their control actions (disabled events) represented in boxes linked to corresponding state. Thus, for the three local supervisors we compute the control action, assigning to each state a set of events that must be disabled, that is, events that at the corresponding state may occur in the respective local plant and are not allowed by the supervisors.

As can be seen in Figure 16, each supervisor disables a set of controllable events according to its states. Notice that two controllable events are associated to activities A and E: events *start* $A$(*start* $E$) and *repair* $A$(*repair* $E$). Thus, supervisor $Sr_2$ can disable these events to meet the

specification $E_2$. In the case activities A and E suspend simultaneously (state 1 of supervisor $Sr_2$ is reached), the supervisor $Sr_2$ always disables *repair E*, since repair of activity A has priority over repair of activity E. Note that no control action is taken on events *start A* and *start E* (none of the supervisors disable these events). In states 0 of the local supervisor $Sr_1$, users can perform activity C (event *start C* disabled in initial state) only after finished activity B (after event *complete B*). Supervisor $Sr_3$ imposes mutual exclusion between activities B and D. If one of this activity happens (occurrence of event *start B* or *start C*), immediately the another activity is disabled (through disabling event *start B* or *start C* in state 1).

Under the point of view of the process, it is necessary that the employees perform all activities, according to the sequence established by supervisors. So, this sequence end when the marked states of the three supervisors are reached and after all activities has been performed.

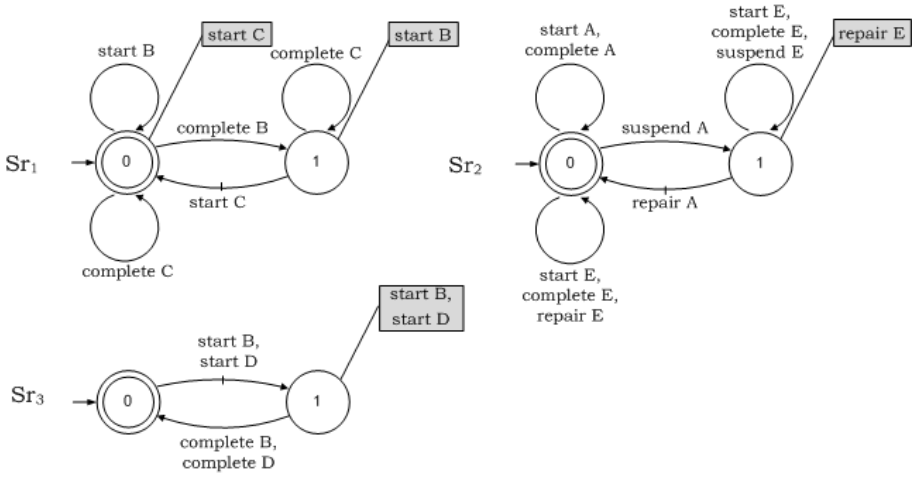


Figure 16: Automata representing reduced local supervisors

We point out that applying the monolithic approach would result in a single global supervisor (non-reduced) with 90 states and 401 transitions. Our corresponding reduced representation of it has only 8 states and 68 transitions. Despite the fact that applying the monolithic approach is this case seems to be feasible (in the sense the reduced monolithic supervisor has a few number of states), in large-scale processes a modular approach is obviously advantageous.

## 5.2 The SCS architecture

To guide the physical implementation of the control system from abstract supervisors, we propose a three level program structure that plays the set of reduced supervisors concurrently (in the case of modular control approach), commands the evolution of asynchronous task models and acts as an interface between the theoretical model and real control event logs.

In our framework, the set of tasks associated to a PAIS plays the same role that the plant in SCT. It means the control action imposed by SCS will be at PAIS activities. Thus, each task belonging to this set is represented by an automaton $G_i$. Also, the global model of this set is represented by automaton $G$, obtained by parallel composition of automata $G_i$. In order to develop the proposed approach, we define the SCS Architecture (SCSA) according to Figure 17. This SCSA assumes the tasks related to a PAIS are represented by a Product System Representation (PSR) as $G_i|i \in I$.

Supervisor's synthesis is accomplished applying the Local Modular Control (LMC) approach, which results in a set of local supervisors as $S_j|j \in J$ (see section 3.5). It is also possible to perform the supervisor's synthesis under a monolithic approach; in this case a global supervisor takes place of the set of local supervisors. This CA is structured as follows: i) The *Modular Supervisors* (MS) level consists of a set of local supervisors, or in a particular case, by a single global supervisor; ii) The *Product System* (PS) level is constituted by a set of structures named product system modules, each of them uniquely associated with an automaton in the PSR chosen to represent an activity; iii) The *Communication Layer* (CL) level is basically a program that has some functionality and manages the communication between process engine of PAIS and PS level.

The communication among the different levels of SCS is depicted in Figure 17. Each supervisor follows the sequence of events (named process events) treated at PS level. Based on such sequence the set of supervisors disables events, which are informed to PS level. The disabling of a controllable event in MS level results in a disabled event at PS level. A product system module associated with a particular automaton in PSR implements the independent behavior of an activity modeled as the corresponding $G_i$, $i \in I$. As we mentioned before, automaton $G_i$ represents the semantics of a task. Each module is responsible for generating controllable and uncontrollable events in the corresponding alphabet $G_i$. In PS level, these events are triggered according to signals received from CL level. The CL level has two main objectives: (i) it is responsible to receive the event log from process engine, to process this log, and to send trigger signals to PS corresponding to controllable and uncontrollable events; (ii) it is responsible to receive the disabled events from PS level, and to send the associated disabled tasks to process engine.

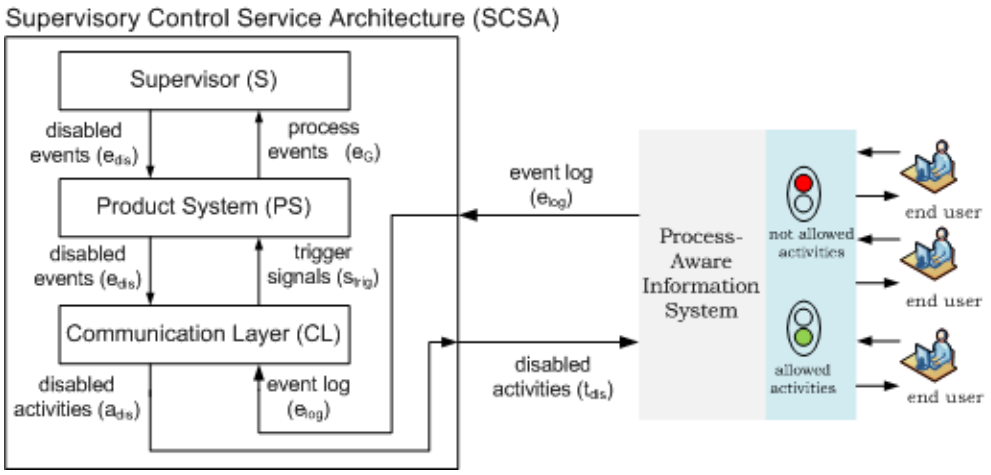


Figure 17: Supervisory control service architecture

The CL level triggers the occurrence of controllable and uncontrollable events at the PS level, according to the on line reading of event log obtained from process engine. To do that, the CL level has to extract the event type from the log and associate it to a trigger signal. Remark that an event log contains a lot of information about activities, but only the information related to events used in modeling of tasks are relevant. Thus, CL level has to extract the event type related to a task. For instance, an event type is associated to a start or a completion of a task. CL level assigns to each event type a trigger signal and each trigger signal is assigned to a controllable or

uncontrollable event (process events). In addition, CL level detects a disabled event from PS level and send it to the process engine the associated disabled task. Thus, the SCS drives the end user of PAIS by sending continuously information about the disabled activities. The end user has to choose an activity (possibly among other) that is not disabled by the SCS.

Figure 18 shows an overview of SCS architecture and its corresponding automata that have been used in Section 5. At the beginning of processing, all modular supervisors and the five activity models are in initial state at MS and PS level, respectively. At this state, it is verified the disabled events and the output map is updated (the set of all disabled events by the three supervisors). Remark that if at least one local supervisor in the set $Sr_j|j \in 1,2,3$ disables the occurrence of a controllable event, then the occurrence of this event at PS level is disabled. A controllable disabled event at MS level is informed to PS level. At this level, this event is not allowed to occur and a new reading is expected from CL level. When the occurrence of an event at PS level is detected, the corresponding product system module executes its transition function and this event is informed to MS level. Then, the active state of some modular supervisor (which has this event in its alphabet) and its corresponding output map is updated. CL level works managing the input and output events from/to PAIS. It is responsible to inform PAIS users which activities are not allowed to occur after an execution sequence of events.

### 5.3 Checking allowed sequence activities

We show in this section some sequences that can be executed by users of PAIS under control of SCS. One way to do this is to replay events in the three level program structure shown in Figure 17. Doing this, it is possible to identify the activities that users can execute according to the control action from SCS. Also, we can check if the overall system behaves well considering the defined business rules. In the case of example show in Section 5, we want to verify if the three business rules shown in Figure 13 are being obeyed.

To initialize the replay of events in the program structure, we consider that the three local supervisors shown in Figure 16 are in initial state when the SCS is turned on. Also, the five activities are enabled (users can choose some execution sequence) in initial states of these supervisors. As shown in Figure 16, the only event being disabled is *start C* by supervisor $Sr_1$. Thus, SCS informs to PAIS that the only activity that users can not execute is activity C. The PAIS users can choose among the other four activities to initiate some execution sequence. Following the control action imposed by the three local supervisors we can identify some activities sequences that users can execute.

Figures 19 to 21 show three examples of execution sequence of activities that are allowed by SCS. Each execution sequence is presented as a state machine. In addition, in all states of each state machine is shown which events are being disabled (see a box attached to every state). Figures 19 and 20 show execution sequences in which users finalize an activity before initiate another one. Notice that after an occurrence of event *start i* ($i = A, B, C, D, E$), the next occurrence will be *complete i*. Thus, it is possible to identify the sequence that has been chosen: Seq 1 ($A \rightarrow E \rightarrow B \rightarrow C \rightarrow D$) and Seq 2 ($B \rightarrow D \rightarrow C \rightarrow A \rightarrow E$). Figure 21 shows other sequence type: after an occurrence

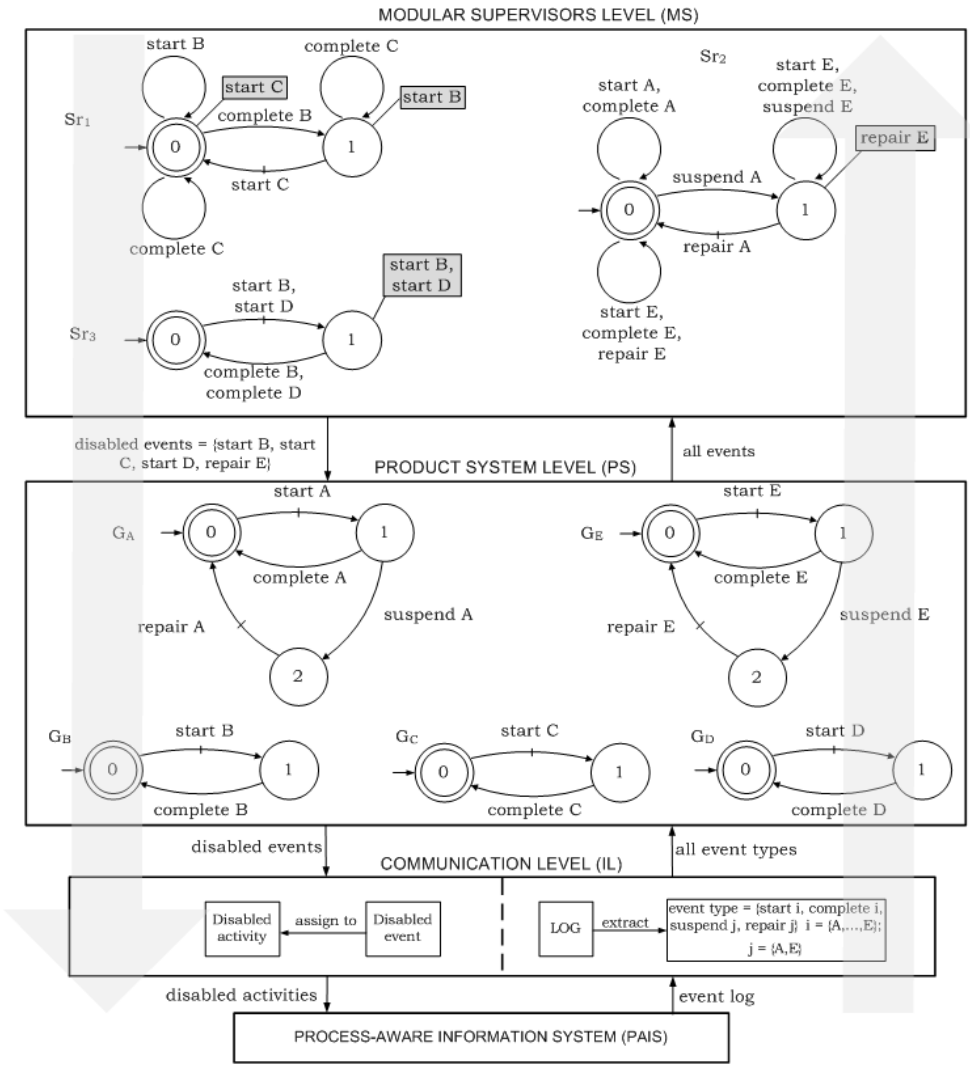


Figure 18: Supervisory control service architecture

of an event *starti*, the next occurrence might be an event *start* $j$ ($i \neq j$) or an event *complete* $i$. Thus, this sequence represents activities being performed simultaneously. For instance, notice that activities A, D and E are initiate before one of them has been completed.

The examples of execution sequences show in Figures 19 to 21 point out the benefit in using SCS. Users can adopt this service as a guide to execute his activities with the guarantee that the business rules or goals are met. Also, the SCS gives flexibility to users in choosing execution sequences. As presented in Figure 21, it is even possible to users execute activities simultaneously, with no violation of pre-defined rules.

# 6 Supervisory control service in ProM

According to the concept of SCS discussed before, this service monitors continuously relevant activities of a PAIS. Thus, we consider SCS as a separate system coupled to the process engine of PAIS. The SCS independently checks if the activities are being performed according to pre-established control action. The purpose is to implement the SCS using the Process Mining Framework ProM. ProM is a pluggable framework that provides a wide variety of plug-ins to extract information about

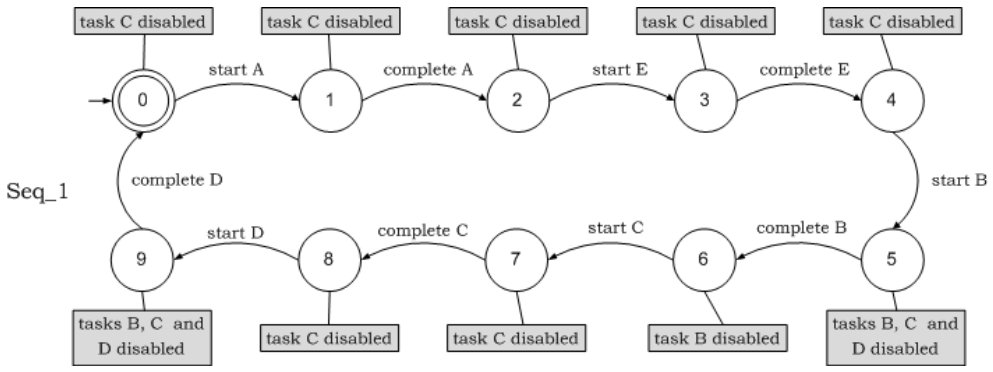


Figure 19: Execution sequence Seq 2 allowed by SCS

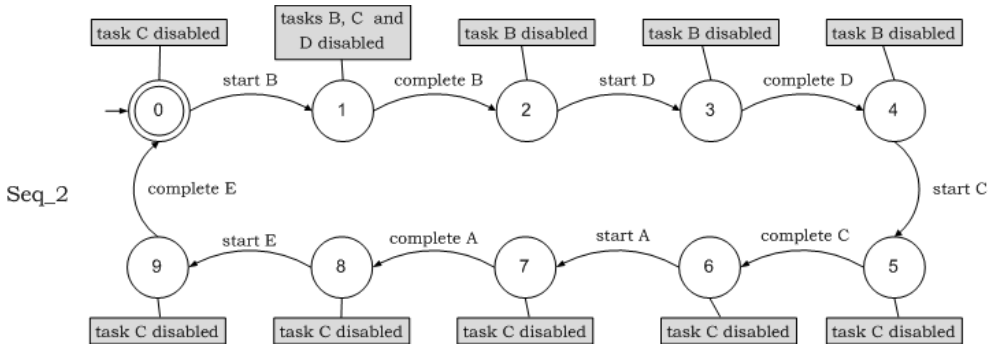


Figure 20: Execution sequence Seq 2 allowed by SCS

a process from event logs (van Dongen et al. 2005), e.g., a process model, an organizational model or decision point information can be discovered. Similar to the recommendation service proposed by Schonenberg et al. (2008), we need to make several extensions to ProM considering that, in contrast to other plug-ins, it is not a posteriori mining technique. The SCS is a service that runs in real-time during process execution.

Our SCS is implemented as a provider for the Operational Support (OS) service in ProM (i.e., OS provider). The OS service is a plug-in of the ProM. The OS provides a TCP/IP interface for communication of the ProM with the external PAIS. The PAIS can request information from the OS service at any point during the execution of a process instance. The request must contain (1) the history of the process instance in the form of the list of activities that have been executed until the moment of request, and (2) the list of activities that are enabled at the moment of request. The general overview of implementation of SCS as a OS provider in ProM is shown in Figure 22. The OS can have connections to an arbitrary number of providers that implement various types of operational support (e.g., 'A', 'B', 'C', 'supervisory control' in Figure 22). When the OS Service receives a request from the PAIS, the OS forwards this request to all connected providers, and sends their responses back to the PAIS. This allows for various types of operational support at the same time. For example, the supervisory control provider may respond by sending the list of disabled activities, some provider for monitoring business rules can send the list of satisified/violated rules, some provide for conformance checking can tell whether the process instance under consideration is conforming to a certain process model, etc.

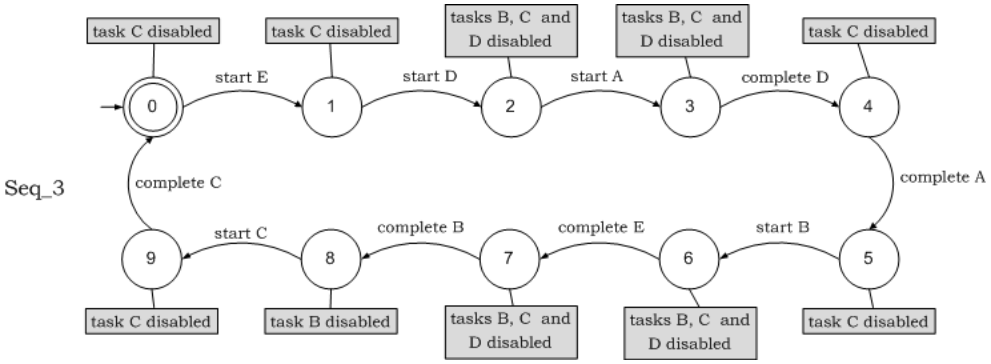


Figure 21: Execution sequence Seq 3 allowed by SCS

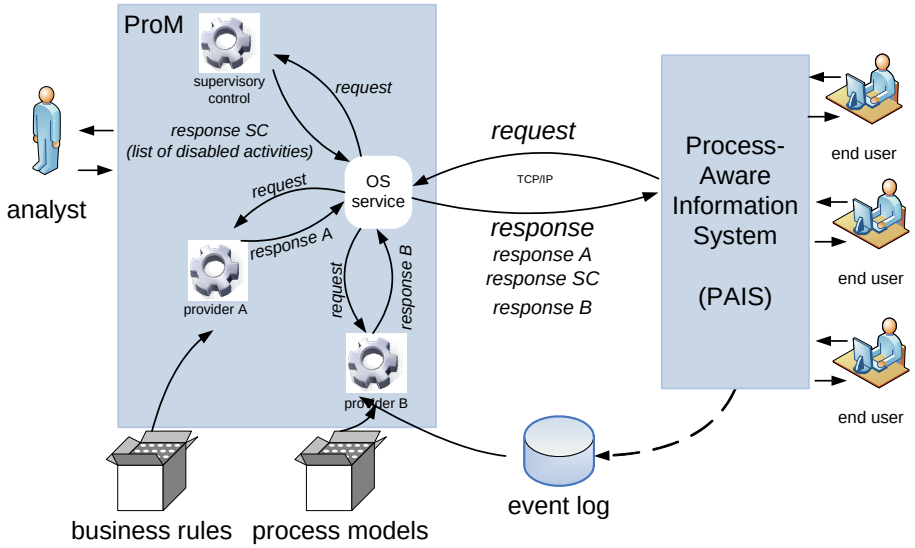


Figure 22: SCS as a OS provider for ProM

## 6.1 Simulation of SCS

To evaluate the correctness of the SCS we conducted a simulation. In our simulation we use the SCS to support the business process shown in Section 5. The process has five tasks (A,B,C,D,E) that have to be executed exactly once and can be executed in any order. However, three business rules (specifications) must be followed by users in such way the executing sequences are restricted. To do that, the SCS informs users which tasks are not allowed to occur after obtaining an observed event sequence from the PAIS. This way the specifications are met. Basically the goal of the experiment is to check whether the right activities are disabled. In addition, it is necessary to check if the supervisors give the expected result concerning the established business rules. The matter here is that specifications represented by automata could not meet correctly the desired business rules (initially established as informal statements). In this case, it is necessary to verify if some allowed tasks sequence does not comply with some of the specifications.

The SCS has been implemented in ProM and the YAWL environment was chosen as the implementation platform to simulate a PAIS. YAWL provides a very powerful and expressive workflow language based on the workflow patterns identified in Van Der Aalst et al. (2003) (Van der Aalst and Ter Hofstede 2005). It also provides a workflow enactment engine (van der Aalst et al. 2004), and an editor for process model creation, that support the control flow, data and (basic) resource

perspectives (Adams et al. 2006). The YAWL environment is open-source and has a service-oriented architecture, allowing the SCS to be developed as a service independent to the engine.

Figure 23 illustrates the architecture of experiment to test the SCS. Since the initial set-up of YAWL was based on the service-oriented architecture, the existing interfaces could be used to also connect to the SCS. Using the web-services paradigm, the SCS can be integrated as a service in YAWL. As the SCS informs users continuously which activities can not be executed, we built an interface in YAWL that implements this functionality. Whenever an user has to choose an execution sequence of activities, she will have available in YAWL environment the information from SCS about that. Thus, this implementation respect the behavior of SCS action: it do not force users to take particular actions. Instead, it gives advice based on business rules that must be respected.

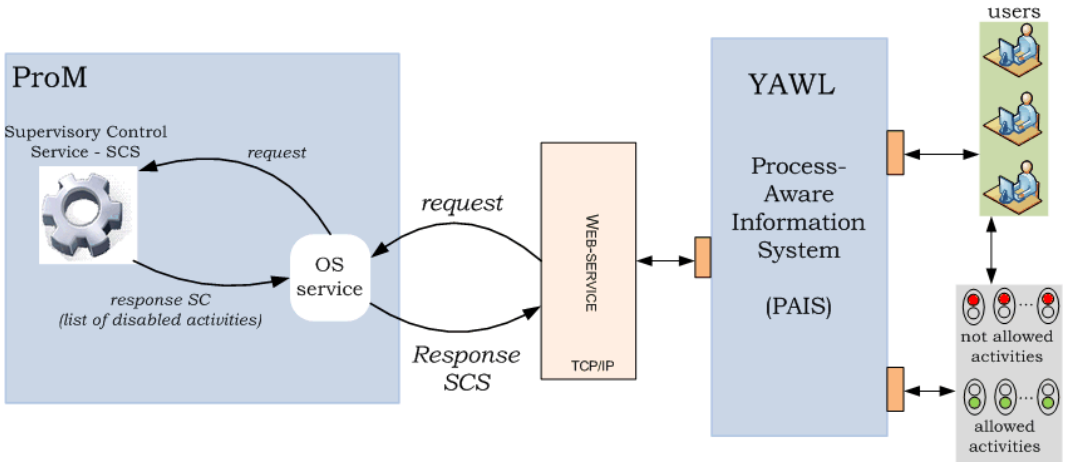


Figure 23: Simulation using SCS as a service for YAWL

The process model implemented in YAWL is shown in figure 24. The process model was discussed in Section 5. One important point is that the five activities can be executed independently. According to Figure 24, an activity labeled "Call SCS guide online" has to be executed just before enabling the five independent activities. Such activity aims to perform the communication between SCS and YAWL. The execution of this activity starts the SCS routines, and the monitoring and control of the further activities can be done. Figure 25 shows the work item available at the moment this activity is enabled. Thus, when user selects this work item the communication between YAWL and the SCS is established. From this point users can choose an execution sequence of activities according to the control action of the SCS.

The execution sequence of these activities can be done according to users decision. The SCS supports users in such way that some business rules are obeyed. Thus, during the execution of activities, the SCS has to inform to YAWL which activities users are not allow to perform (or which activities can happen next). The control action sent to YAWL is a list assigning each activity to a Boolean value. When an activity is assigned to false, the SCS is disabling the event that corresponds to the beginning of such activity. Otherwise, the SCS allows activities to be executed in YAWL. As long as an event occurs in YAWL, the program in the SCS is updated and a new control action is sent.

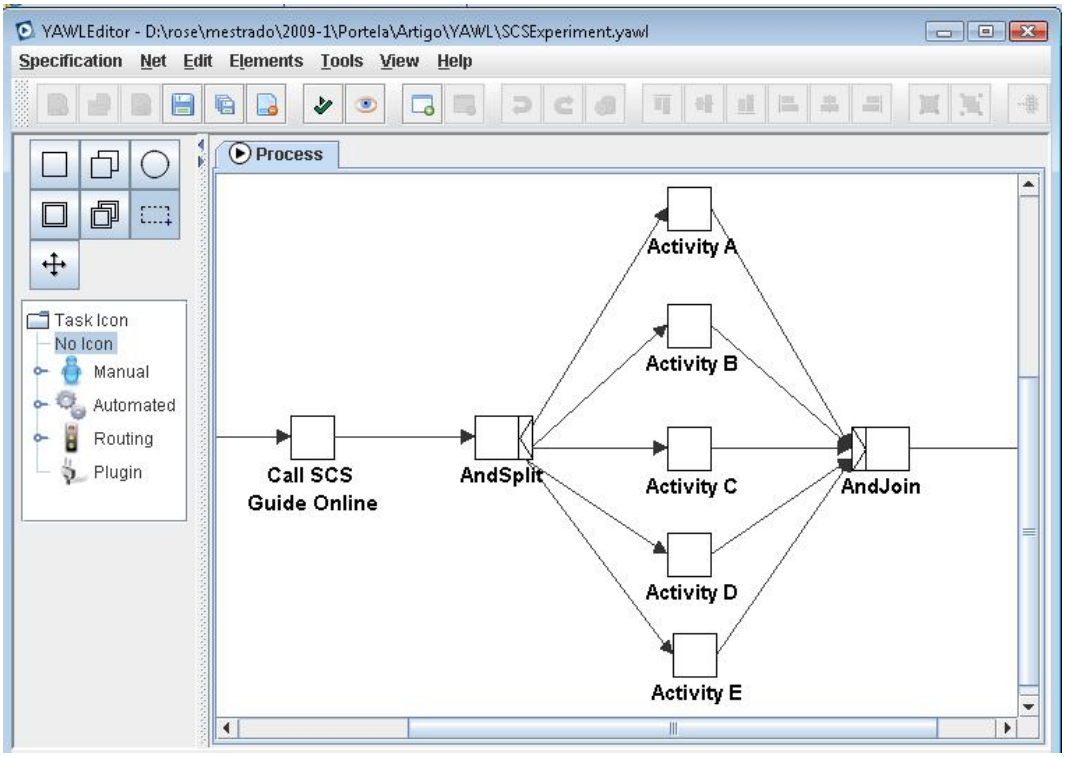


Figure 24: Model of business process in YAWL editor

Figures 26 and 27 show the interplay between SCS (in ProM) and YAWL. In ProM, when the communication is established, SCS sends the first control action according to specified in the initial states of the local supervisors (see Figure 16). Notice that in ProM the control action is sent as a list of disabled (and enabled) activities. According to Figure 26, the only disabled activity is the activity C (for the case 37). In YAWL we built an interface that that provides users the information (disabled activities) from the SCS. As users select a specific work item, such interface shows up informing which activities are disabled. A semaphore assigned to each activity indicates the disabling of activities (red light). Also, the interface indicates which event has been sent to SCS. For example, the interface shown in Figure 27 appears just after the work item *Call SCS Guide Online* has been selected. According to this figure, the sent event was *none*, indicating that no event has been sent until that moment. Notice that no activity (among activities A,B,C,D and E) has been started yet. According to Figure 27, the first information available to users is that activity C can not be executed (see the red light assigned to it) at that moment.

Figures 28 to 30 show an example in which a execution sequence is initiated. Figure 28 shows that after execution of *Call SCS online guide*, users choose which activity can be executed next (note that activity C is blocked and it does not appear as a work item available). For example, a user selecting activity B to be executed, immediately the event *start B*) is sent to SCS and it updates its control action. Thus, the new control action after the occurrence of event *start B* can be seen in Figure 29: the activities B, C and D are disabled. Figure 30 shows the interface in YAWL informing users that activities B, C and D are disabled by the SCS after occurrence of event *start B* (see the red light assigned to these activities). At this moment a user will execute the activity B until the end or she also can check the work items available in order to start the

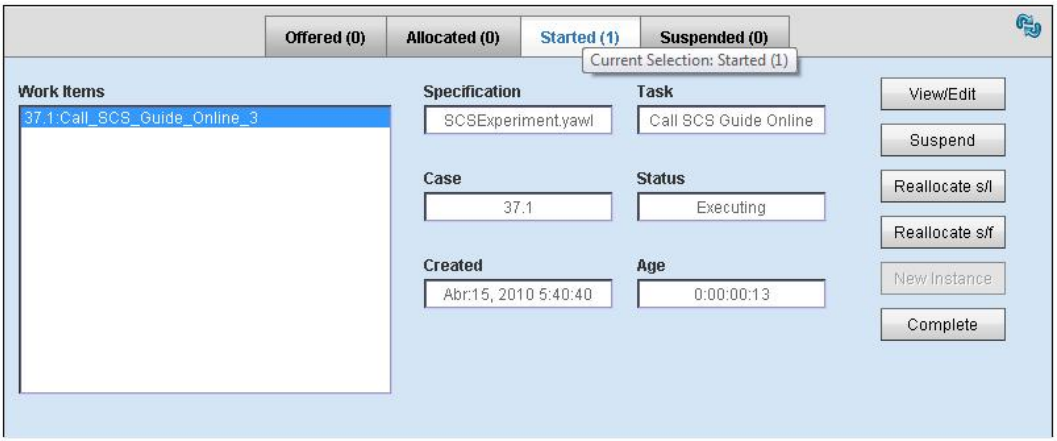


Figure 25: Work item available - SCS guide online

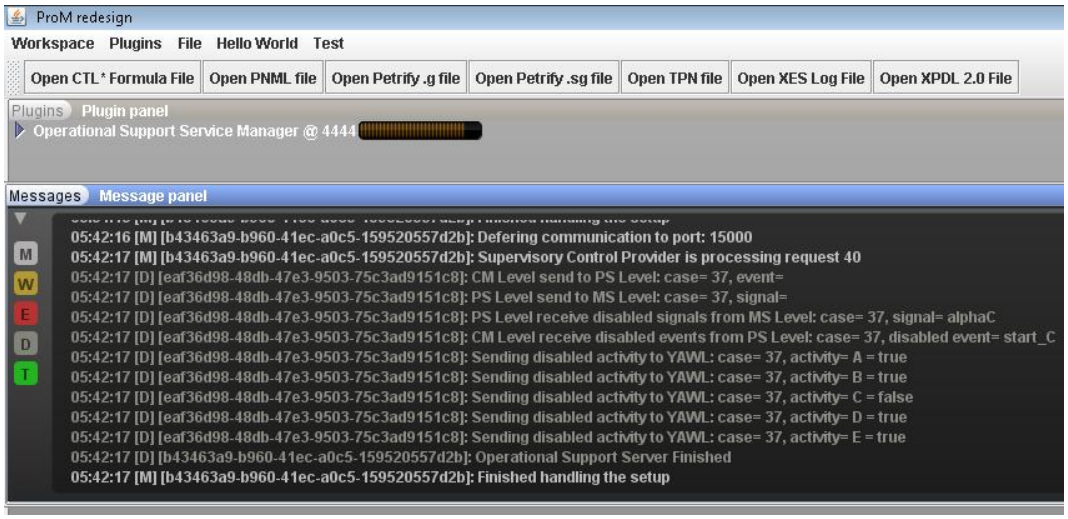


Figure 26: The first control action of SCS in ProM

execution of another activity.

Finalizing activity B (signalized through an occurrence of event *complete B*), the SCS updates its control action and send it to YAWL. Figure 31 shows the new control action after the event *complete B* has been received. Notice that only activity B is disabled by SCS. Figure 33 shows the interface in YAWL after receive the new control action from SCS. Notice that only the activity B is assigned to a red light, indicating that it is not allowed to occur. According to Figure 32 a new set of work items is available after finalization of activity B. In this way, completing the execution sequence based on control action of the SCS, users have the guarantee that no violation has been done. This scenario illustrates the functionality that has been implemented in YAWL.

# 7 Related work

Many approaches has proposed to add control to the business processes and to avoid incorrect or undesirable executions of the activities. A stream of research proposes using rule-based or constraint-based modeling languages. Glance et al. (1996) use process grammars for definition of rules involving activities and documents. The Freeflow prototype (Dourish et al. 1996) uses

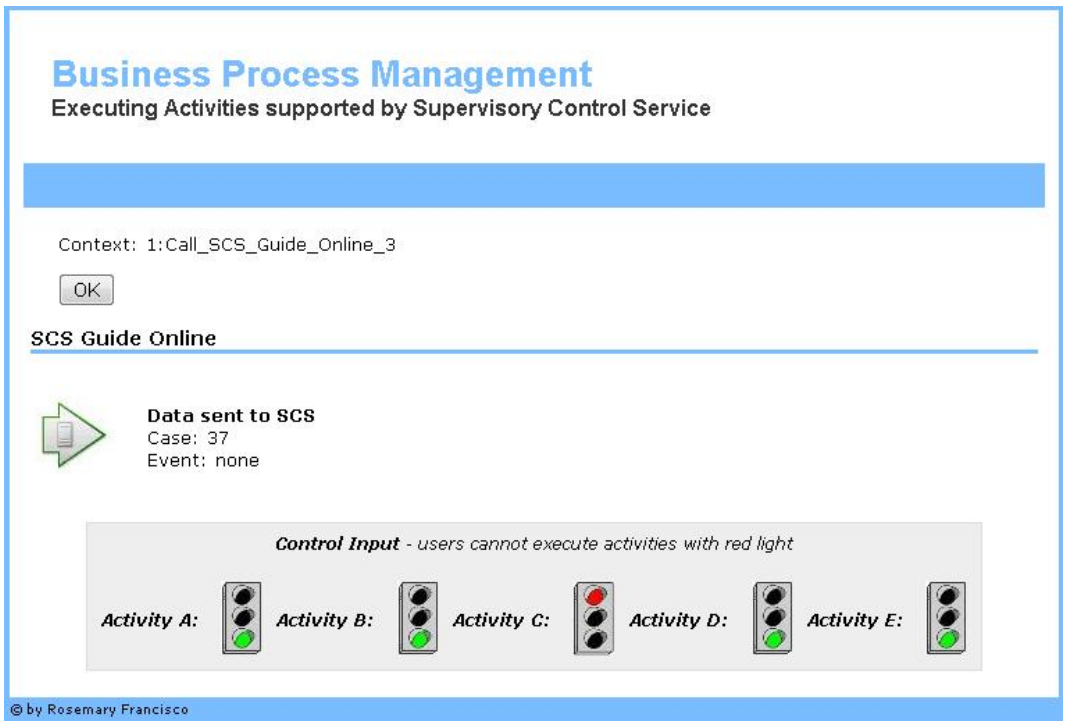


Figure 27: The first control action in YAWL - activity C disabled

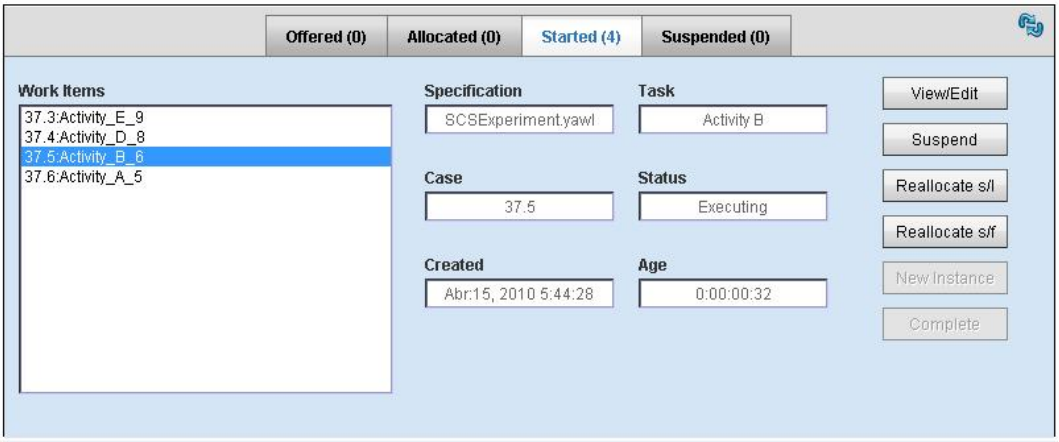


Figure 28: Work items available after *Call SCS Guide Online*

constraints for building declarative process models. Freeflow constraints represent dependencies between states (e.g., inactive, active, disabled, enabled, etc.) of different activities, i.e., an activity can enter a specific state only if another activity is in a certain state. (Wainer and de Lima Bezerra 2003) present a constraint-based language that uses rules involving preconditions (that must hold before an activity can be executed), postconditions (that must hold after an activity is executed) and additional conditions that must hold in general before or after an activity is executed. Joeris (2000) proposes flexible workflow enactment based on event-condition-action (ECA) rules. In (Lu et al. 2006), a temporal constraint network is proposed for business process execution. (Attie et al. 1996) and (Attie et al. 1993) use Computational Tree Logic (CTL) for the specification of inter-task dependencies amongst different unique events (e.g., commit dependency, abort dependency, conditional existence dependency, etc.). Dependencies are transformed into automata, which are used by a central scheduler to decide if particular events are accepted, delayed or rejected. (Raposo

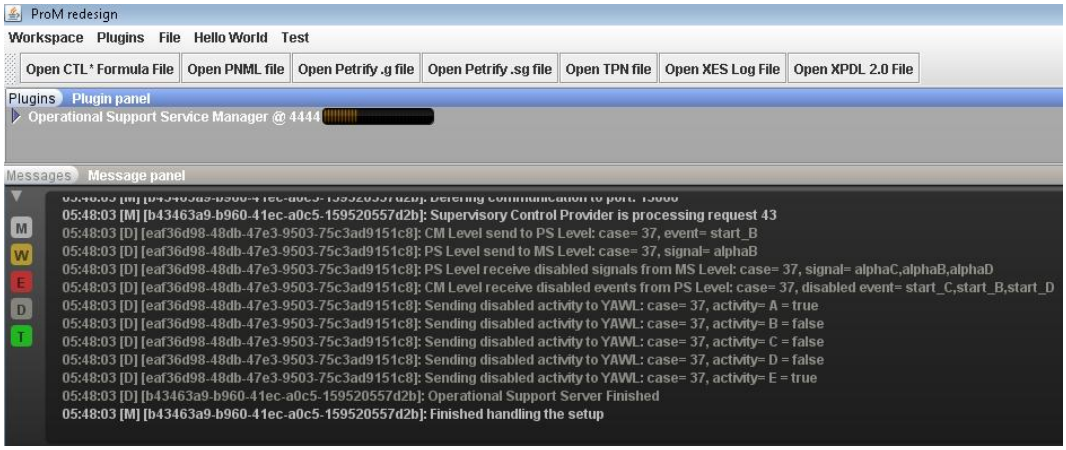


Figure 29: SCS control action in ProM after occurrence initiation of activity B

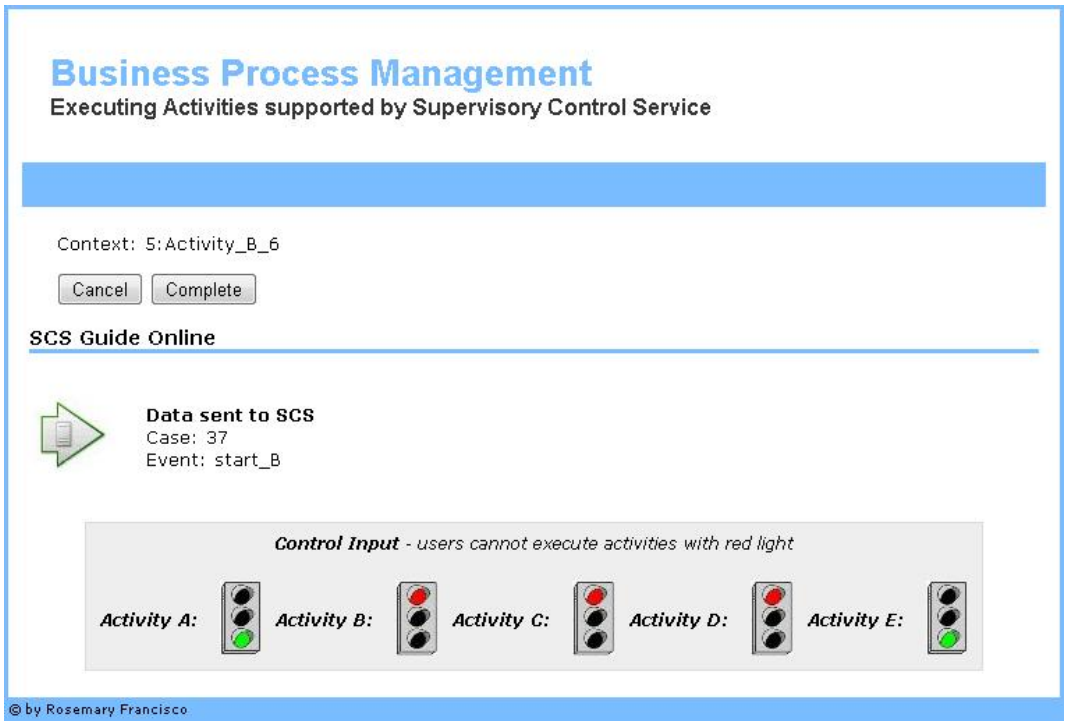


Figure 30: Disabled activities in YAWL after initiation of activity B

and Hugo ) and (Raposo et al. 2001) propose a larger set of basic interdependencies and propose modeling their coordination using Petri nets.

In (van der Aalst et al. 2009) and (Pesic et al. 2007) a constraint-based WFMS called DECLARE is presented. Declare is a framework that implements various declarative languages, e.g., DecSerFlow (van der Aalst and Pesic 2006) and ConDec (Pesic and van der Aalst 2006), and supports flexibility by design, flexibility by deviation, and flexibility by change. In DECLARE the possible ordering of activities are determined by constrains. Thus, any order of execution sequence can be possible as long as constraints are not violated. Everything that does not violate the constraints is allowed. Van der Aalst et al. (2008) use DECLARE in the context of FAAS (Flexibility as a Service). The basic idea of FAAS is to join different notions of flexibility using the concept of a service. To do that, the authors propose that interfaces between engines support different languages.

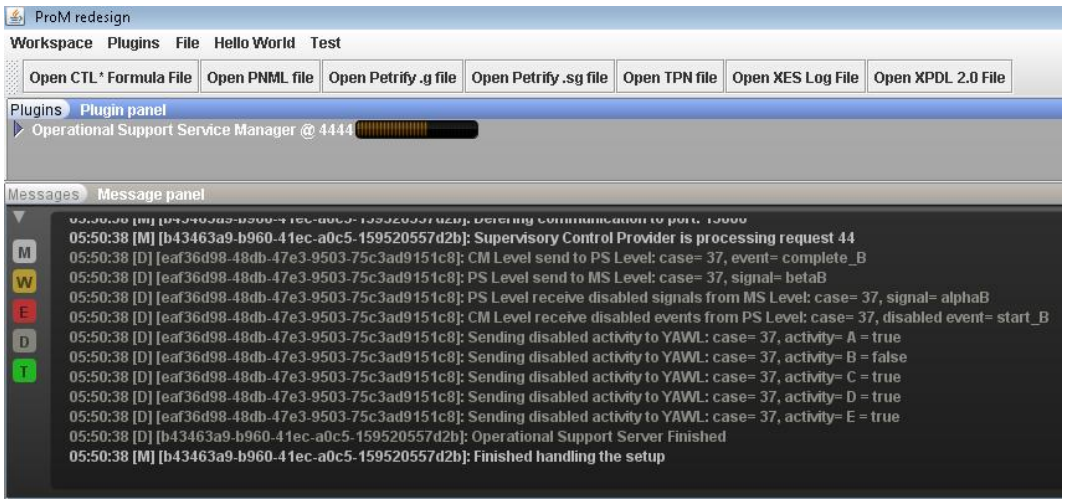


Figure 31: SCS control action in ProM after finalization of activity B

Another stream of research proposes to use actual data from PAISs to support decision making. In (Rozinat et al. 2009) process mining and simulation techniques are combined in the context of the YAWL system to accurately predict potential near-future behaviors for different scenarios. In (Dongen et al. 2008) non-parametric regression is used to predict completion times. A recommendation service that uses historic information for guiding the user to select the next work item has been implemented in ProM (Schonenberg et al. 2008), which is similar to the case-based reasoning presented in (Weber et al. 2004). A recommender for execution of business processes based on the Product Data Model (PDM) is presented in (Vanderfeesten et al. 2008a). Other works propose to use the concept of Adaptive PAIS, in which users are able to make structural process changes both the handling of exceptions and the evolution of business processes. ProCycle Rinderle et al. (2005), Weber et al. (2009) and Minor et al. (2008) present approaches that support users to conduct instance specific changes through change reuse. Worklets (Adams et al. 2006) and Pockets of Flexibility (Sadiq et al. 2005) approaches provide user assistance by providing simple support for the reuse of previously selected or defined strategies. Vanderfeesten et al. (2008b) propose recommendations based on product data model to select the step, which meets the performance goals of the process best (e.g., lowest cost, shortest remaining cycle time).

Run time support is an important research topic in the context of world wide web. Some examples are approaches based on business rules (Lazovik et al. 2004), BPEL (Baresi et al. 2004), event calculus (Mahbub and Spanoudakis 2004), etc. Recommender systems that support users in their decision-making based on the user's preferences and are becoming an essential part of e-commerce and information seeking activities (Resnick and Varian 1997). van der Aalst et al. (2010) provide a framework for positioning the various types of process mining and details the aspect of operational support for running processes in a generic manner. Time-based operational support can be used to detect deadline violations, predict the remaining processing time, and recommend activities that minimize flow times.

Related work in the context of Supervisory Control Theory addresses implementation of its control structure. Publications that deal with this issue and academic test bed applications are presented in (Balemi et al. 1993) (Leduc 1996) (Brandin 1996) (Lauzon et al. 1997) (Fabian and

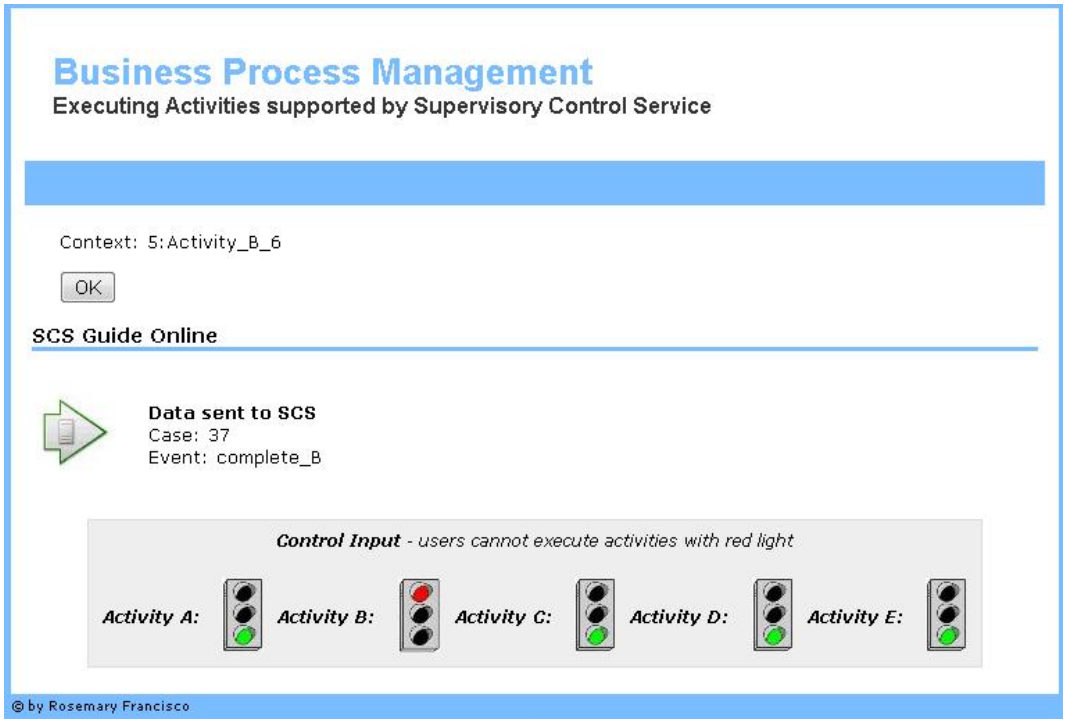


Figure 32: Disabled activities in YAWL after finalization of activity B

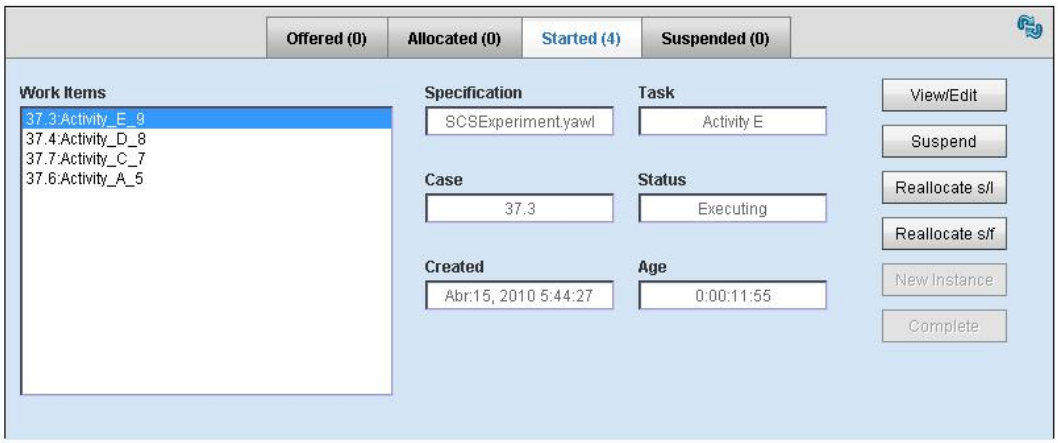


Figure 33: Work items available after finalization of activity B

Hellgren 1998) (Charbonnier et al. 1999) (Dietrich et al. 2002) (Liu and Darabi 2002) (Malik 2002) (De Queiroz and Cury 2002) (Music and Matko 2002) (Gouyon et al. 2004) (Mušic et al. 2005) (Cote et al. 2005) (Vieira et al. 2006) (Uzam and Wonham 2006) (Li and Jiang 2007) (Silva et al. 2007) (Silva et al. 2008).

# 8 Conclusion

This paper has addressed the implementation of a supervisory control service in order to support and restrict the execution of activities in Process-Aware Information Systems (PAIS). Flexible processes provide much freedom for users. A user might choose, according to her own decision, a execution sequence of activities. However, we have no guarantees that the chosen sequence is the best one or at least it conforms to some established business rules. In this context, the SCS

proposed in this paper aims to monitor and restrict the execution sequence of activities in such way that business rules are always obeyed. By using our SCS approach, we aim at offering support based on previous business rules but not limit the user by imposing rigid control-flow structures. In fact, the SCS informs users which activities are not allowed after an observed trace of events at run-time. The users continue with some level of freedom, because they can choose execution sequences that are allowed under the supervision of SCS.

Many approaches deal with the modeling of Discrete Event Systems and the synthesis of its controllers. To build the engine of SCS we use the approach proposed by Ramadge and Wonham (1989). Supervisory Control Theory (SCT) is suited for the control design of DES when restrictions need to be imposed. Based upon a model that describes the free behavior of the system to be controlled and a set of specifications, it is possible to perform a formal synthesis of a supervisor. The control action of the supervisor restricts the behavior of the system to be controlled, so that all requirements will be satisfied. For performance reasons, the local modular approach (De Queiroz and Cury 2000b) was used to synthesize local supervisors instead of a unique supervisor.

The SCS has been implemented by extending ProM and the experiment that has been performed to show its feasibility. The computational infra-structure already implemented in ProM allows communication with external applications and allows the SCS to be integrated with PAISs that records events. In this paper we demonstrated that the SCS in ProM can cooperate with the YAWL system. Future work will aim at extending the application and implementation of SCS in large-scale business processes. Through a large number of case studies we intent to make SCS a tool for decision making support. Moreover, we plan to incorporate other control approaches and investigate the suitability of the approaches in particular settings.

# References


Adams, M., A. Hofstede, D. Edmond, and W. van der Aalst (2006). Worklets: A service-oriented implementation of dynamic flexibility in workflows. *Lecture Notes in Computer Science 4275*, 291.

Attie, P., M. Singh, E. Emerson, A. Sheth, and M. Rusinkiewicz (1996). Scheduling workflows by enforcing intertask dependencies. *Distributed Systems Engineering 3*, 222–238.

Attie, P., M. Singh, A. Sheth, and M. Rusinkiewicz (1993). Specifying and enforcing intertask dependencies. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 134–134. Citeseer.

Balemi, S., G. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. Franklin (1993). Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control 38*(7), 1040–1059.

Baresi, L., C. Ghezzi, and S. Guinea (2004). Smart Monitors for Composed Services. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, NY, USA, pp. 193–202. ACM Press.

Brandin, B. (1996). The real-time supervisory control of an experimental manufacturing cell. *IEEE Transactions on robotics and automation 12*(1), 1–14.

Carroll, J. and D. Long (1989). *Theory of finite automata with an introduction to formal languages*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Cassandras, C. and S. Lafortune (2008). *Introduction to discrete event systems*. Springer Verlag.

Charbonnier, F., H. Alla, and R. David (1999). The supervised control of discrete-event dynamic systems. *IEEE Transactions on Control Systems Technology 7*(2), 175.

Chen, Y., S. Lafortune, and F. Lin (1995). Modular supervisory control with priorities for discrete event systems. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, Volume 1.

Cote, D., R. St-Denis, and S. Kerjean (2005). Generative programming for programmable logic controllers. In *10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005*, Volume 2.

Cury, J. and J. Eyzell (2001). Exploiting symmetry in the synthesis of supervisors for discrete event systems. *IEEE Transactions on Automatic Control 46*(9), 1500–1505.

De Queiroz, M. and J. Cury (2000a). Modular control of composed systems. In *Proceedings of the American Control Conference*, Volume 6, pp. 4051–4055.

De Queiroz, M. and J. Cury (2000b). Modular supervisory control of large scale discrete event systems. *Discrete Event Systems: Analysis and Control*, 103–110.

De Queiroz, M. and J. Cury (2002). Synthesis and implementation of local modular supervisory control for a manufacturing cell. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*, pp. 377–382. Citeseer.

Dietrich, P., R. Malik, W. Wonham, and B. Brandin (2002). Implementation considerations in supervisory control. *Synthesis and control of discrete event systems 1*.

Dongen, B., R. Crooy, and W. Aalst (2008). Cycle Time Prediction: When Will This Case Finally Be Finished? In R. Meersman and Z. Tari (Eds.), *Proceedings of the 16th International Conference on Cooperative Information Systems, CoopIS 2008, OTM 2008, Part I*, Volume 5331 of *Lecture Notes in Computer Science*, pp. 319–336. Springer-Verlag, Berlin.

Dourish, P., J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw (1996). Freeflow: mediating between representation and action in workflow systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pp. 190–198. ACM New York, NY, USA.

Dumas, M., W. van der Aalst, and A. Ter Hofstede (2005). *Process-aware information systems: bridging people and software through process technology*. Wiley-Blackwell.

Fabian, M. and A. Hellgren (1998). PLC-based implementation of supervisory control for discrete eventsystems. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, Volume 3.

Feng, L. and W. Wonham (2006). TCT: A computation tool for supervisory control synthesis. In *International Workshop on Discrete Event Systems*, pp. 388–389.

Glance, N., D. Pagani, and R. Pareschi (1996). Generalized process structure grammars GPSG for flexible representations of work. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pp. 180–189. ACM New York, NY, USA.

Gouyon, D., J. Petin, and A. Gouin (2004). Pragmatic approach for modular control synthesis and implementation. *International Journal of Production Research 42*(14), 2839–2858.

Hopcroft, J., R. Motwani, and J. Ullman (2006). *Introduction to automata theory, languages, and computation*. Addison-wesley.

Joeris, G. (2000). Decentralized and flexible workflow enactment based on task coordination agents. In *Agent-Oriented Information Systems 2000.: Proceedings of AOIS-2000 at CAiSE* 00.*, pp. 41. BoD–Books on Demand.

Lauzon, S., J. Mills, and B. Benhabib (1997). An implementation methodology for the supervisory control of flexible manufacturing workcells. *Journal of Manufacturing Systems 16*(2), 91–101.

Lazovik, A., M. Aiello, and M. Papazoglou (2004). Associating Assertions with Business Processes and Monitoring their Execution. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, NY, USA, pp. 94–104. ACM Press.

Leduc, R. (1996). *PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective*. Ph. D. thesis, University of Toronto.

Li, L. and Z. Jiang (2007). A hybrid supervisory control approach for virtual production systems. *The International Journal of Advanced Manufacturing Technology 32*(9), 1033–1044.

Liu, J. and H. Darabi (2002). Ladder logic implementation of Ramadge-Wonham supervisory controller. In *Proceedings of 6th Int. Workshop on Discrete Event Systems*, Volume 1, pp. 383–389.

Lu, R., S. Sadiq, V. Padmanabhan, and G. Governatori (2006). Using a temporal constraint network for business process execution. In *Proceedings of the 17th Australasian Database Conference-Volume 49*, pp. 166. Australian Computer Society, Inc.

Mahbub, K. and G. Spanoudakis (2004). A Framework for Requirents Monitoring of Service Based Systems. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, NY, USA, pp. 84–93. ACM Press.

Malik, P. (2002). Generating controllers from discrete-event models. In *Proceedings of Modelling and Verifying Parallel Processes*, Volume 1, pp. 337–342.

Minhas, R. (2002). *Complexity reduction in discrete event systems*. Ph. D. thesis, University of Toronto.

Minor, M., A. Tartakovski, D. Schmalen, and R. Bergmann (2008). Agile workflow technology and case-based change reuse for long-term processes. *International Journal of Intelligent Information Technologies 4*(1), 80–98.

Mušic, G., D. Gradišar, and D. Matko (2005). IEC 61131-3 Compliant Control Code Generation from Discrete Event Models. In *Proceedings of the 13th Mediterranean Conference on Control and Automation*, pp. 346–351.

Music, G. and D. Matko (2002). Discrete event control theory applied to PLC programming. *Automatica 43*(1-2), 21–28.

Pesic, M., M. Schonenberg, N. Sidorova, and W. van der Aalst (2007). Constraint-based workflow models: Change made easy. *Lecture Notes in Computer Science 4803*, 77.

Pesic, M. and W. van der Aalst (2006). A declarative approach for flexible business processes management. *Lecture Notes in Computer Science 4103*, 169.

Queiroz, M., J. Cury, and W. Wonham (2005). Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems 15*(4), 375–395.

Ramadge, P. and W. Wonham (1987). Supervisory control of a class of discrete event systems. *Siam Journal of Manufacturing Systems Control and Optimization 25*(1), 206–230.

Ramadge, P. and W. Wonham (1988). Modular supervisory control of discrete event systems. *Mathematics of Control, Signal and Systems 1*(1), 13–30.

Ramadge, P. and W. Wonham (1989). The control of discrete event systems. *Proceedings of the IEEE 77*(1), 81–98.

Raposo, A. and F. Hugo. Defining task interdependencies and coordination mechanisms for collaborative systems. *Cooperative systems design: a challenge of the mobility age*, 88–103.

Raposo, A., L. Magalhães, I. Ricarte, and H. Fuks (2001). Coordination of collaborative activities: A framework for the definition of tasks interdependencies. In *7th International Workshop on Groupware-CRIWG*, Volume 2001.

Resnick, P. and H. Varian (1997). Recommender systems. *Communications of the ACM 40*(3), 56–58.

Rinderle, S., B. Weber, M. Reichert, and W. Wild (2005). Integrating process learning and process evolution-a semantics based approach. *Lecture Notes in Computer Science 3649*, 252.

Rozinat, A. and W. van der Aalst (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems 33*(1), 64–95.

Rozinat, A., M. Wynn, W. Aalst, A. Hofstede, and C. Fidge (2009). Workflow Simulation for Operational Decision Support. *Data and Knowledge Engineering 68*(9), 834–850.

Rudie, K. and W. Wonham (1992). Think globally, act locally: Decentralized supervisory control. *IEEE transactions on automatic control 37*(11), 1692–1708.

Sadiq, S., M. Orlowska, and W. Sadiq (2005). Specification and validation of process constraints for flexible workflows. *Information Systems 30*(5), 349–378.

Schonenberg, H., R. Mans, N. Russell, N. Mulyar, and W. van der Aalst (2008a). Process flexibility: A survey of contemporary approaches. In *Advances in Enterprise Engineering I: 1st International Workshop Ciao! and 4th International Workshop Eomas at Caise 2008, Montpellier, France, June 16-17, 2008, Proceedings*, pp. 16. Springer.

Schonenberg, H., B. Weber, B. van Dongen, and W. van der Aalst (2008). Supporting Flexible Processes through Recommendations Based on History. In *Proceedings of the 6th International Conference on Business Process Management*, pp. 51–66. Springer.

Schonenberg, M., R. Mans, N. Russell, N. Mulyar, and W. van der Aalst (2008b). Towards a taxonomy of process flexibility. In *CAiSE Forum*, pp. 81–84.

Silva, D., E. Santos, A. Vieira, and M. de Paula (2007). Application of the supervisory control theory to automated systems of multi-product manufacturing. In *IEEE Conference on Emerging Technologies and Factory Automation, 2007. ETFA*, pp. 689–696.

Silva, D., E. Santos, A. Vieira, and M. de Paula (2008). Application of the supervisory control theory in the project of a robot-centered, variable routed system controller. In *IEEE International Conference on Emerging Technologies and Factory Automation ( ETFA 2008), 2008.*, pp. 751–758.

Su, R. and W. Wonham (2004). Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems 14*(1), 31–53.

Uzam, M. and W. Wonham (2006). A hybrid approach to supervisory control of discrete event systems coupling RW supervisors to Petri nets. *The International Journal of Advanced Manufacturing Technology 28*(7), 747–760.

Van der Aalst, W., M. Adams, A. ter Hofstede, M. Pesic, and H. Schonenberg (2008). Flexibility as a Service. *BPM Center Report BPM-08-09*.

van der Aalst, W., L. Aldred, M. Dumas, and A. Hofstede (2004). Design and implementation of the YAWL system. *Lecture Notes in Computer Science*, 142–159.

van der Aalst, W. and M. Pesic (2006). DecSerFlow: Towards a truly declarative service flow language. *Lectur Notes in Computer Science 4184*, 1.

van der Aalst, W., M. Pesic, and H. Schonenberg (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development 23*(2), 99–113.

van der Aalst, W., M. Pesic, and M. Song (2010). Beyond Process Mining: From the Past to Present and Future. Technical report, Eindhoven University of Technology, The Netherlands.

van der Aalst, W., M. Rosemann, and M. Dumas (2007). Deadline-based escalation in process-aware information systems. *Decision Support Systems 43*(2), 492–511.

Van der Aalst, W. and A. Ter Hofstede (2005). YAWL: yet another workflow language. *Information Systems 30*(4), 245–275.

Van Der Aalst, W., A. Ter Hofstede, B. Kiepuszewski, and A. Barros (2003). Workflow patterns. *Distributed and parallel databases 14*(1), 5–51.

Van der Aalst, W., B. Van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters (2003). Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering 47*(2), 237–267.

van Dongen, B., A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst (2005). The ProM framework: A new era in process mining tool support. *Application and Theory of Petri Nets 3536*, 444–454.

Vanderfeesten, I., H. Reijers, and W. Aalst (2008a). Product Based Workflow Support: Dynamic Workflow Execution. In Z. Bellahsene and M. Léonard (Eds.), *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08)*, Volume 5074 of *Lecture Notes in Computer Science*, pp. 571–574. Springer-Verlag, Berlin.

Vanderfeesten, I., H. Reijers, and W. Aalst (2008b). Product based workflow support: dynamic workflow execution. *Lecture Notes in Computer Science 5074*, 571–574.

Vaz, A. and W. Wonham (1986). On supervisor reduction in discrete-event systems. *International Journal of Control 44*(2), 475–491.

Vieira, A., J. Cury, and M. de Queiroz (2006). A Model for PLC Implementation of Supervisory Control of Discrete Event Systems. In *IEEE Conference on Emerging Technologies and Factory Automation, 2006. ETFA'06*, pp. 225–232.

Wainer, J. and F. de Lima Bezerra (2003). Constraint-based flexible workflows. *Lecture Notes in Computer Science*, 151–158.

Weber, B., M. Reichert, and S. Rinderle-Ma (2008). Change patterns and change support features–enhancing flexibility in process-aware information systems. *Data & knowledge engineering 66*(3), 438–466.

Weber, B., M. Reichert, S. Rinderle-Ma, and W. Wild (2009). Providing integrated life cycle support in process-aware information systems. *International Journal of Cooperative Information Systems 18*(1), 115–165.

Weber, B., S. Rinderle, and M. Reichert (2007). Change support in process-aware information systems-a pattern-based analysis. Technical report, Citeseer.

Weber, B., W. Wild, and R. Breu (2004). CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In *Advances in Case-Based Reasoning*, Volume 3155 of *Lecture Notes in Computer Science*, pp. 434–448. Springer-Verlag, Berlin.

Wong, K., J. Thistle, H. Hoang, and R. Malhamé (1995). Conflict resolution in modular control with applications to featureinteraction. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, Volume 1.

Wong, K. and W. Wonham (1996). Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems 6*(3), 241–273.

Wong, K. and W. Wonham (1998). Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems 8*(3), 247–297.

Zhong, H. and W. Wonham (1990). On the consistency of hierarchical supervision in discrete-eventsystems. *IEEE Transactions on Automatic Control 35*(10), 1125–1134.