Sauro Schaidt

Variability management in constraint-based processes: contributions to selection of process variants

Curitiba

2017

Sauro Schaidt

Variability management in constraint-based processes: contributions to selection of process variants

Tese apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Doutor em Engenharia de Produção e Sistema, área de concentraccão em Automação e Controle de Sistemas

Pontifícia Universidade Católica do Paraná - PUCPR

Escola Politécnica

Programa de Pós-Graduação em Engenharia de Produção e Sistemas

Orientador: Prof. Dr. Eduardo Alves Portela Santos

Curitiba

2017

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

Schaidt, Sauro

S296v
2017

Viability management in constraint-based processes : contribuitions to selection of process variants / Sauro Schaidt ; orientador, Eduardo Alves Portela Santos. – 2017.
231 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2017
Bibliografia: f. 213-231

1. Controle da produção. 2. Controle de processos. 3. Negócios. 4. Engenharia de produção. I. Santos, Eduardo Alves Portela. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Engenharia de Produção e Sistemas. III. Título

CDD 22. ed. – 658.56

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Engenharia de Produção e Sistemas

PUCPR

TERMO DE APROVAÇÃO

Sauro Schaidt

VARIABILITY MANAGEMENT IN CONSTRAINT-BASED PROCESSES: CONTRIBUTIONS TO SELECTION OF PROCESS VARIANTS.

Tese aprovada como requisito parcial para obtenção do grau de Doutor no Curso de Doutorado em Engenharia de Produção e Sistemas, Programa de Pós-Graduação em Engenharia de Produção e Sistemas, da Escola Politécnica da Pontifícia Universidade Católica do Paraná, pela seguinte banca examinadora:

[illegible]

Presidente da Banca
Prof. Dr. Eduardo Alves Portela Santos
(Orientador)

[illegible]

Prof. Dr. Fernando Deschamps
(Membro Interno)

[illegible]

Prof. Dr. Eduardo de Freitas Rocha Loures
(Membro Externo)

[illegible]

Prof. Dr. Agnelo Denis Vieira
(Membro Externo - PUCPR)

[illegible]

Prof. Dr. Ângelo Márcio Oliveira Sant'Anna
(Membro Externo)

[illegible]

Prof. Dr. José Eduardo Pécora Júnior
(Membro Externo)

Curitiba, 30 de agosto de 2017.

Rua Imaculada Conceição, 1155 - Prado Velho – CEP: 80215-901 - Curitiba - Paraná - Brasil
Tel: +55 41 3271-2579 www.pucpr.br/ppgeps

A Ary e Guida.

Agradecimentos

Ao meu orientador, Prof. Dr. Eduardo Alves Portela Santos, por todo o apoio, acompanhamento e incentivo ao longo desta pesquisa aqui no Basil e durante o meu intercâmbio acadêmico fora do país.

Ao professor Dr. Osiris Canciglieri por todo o esforço e empenho que viabilizaram o meu intercâmbio acadêmico fora do país.

Ao professor Dr. Thomas Hildebrandt por todo a atenção e apoio dispensados durante o tempo do meu intercâmbio acadêmico.

Ao professor Dr. Agnelo Denis Vieira, pelas contribuições com a Teoria de Controle Supervisório.

Ao professor Dr. Fernando Deschamps, pelas contribuições com tópicos de gerenciamento de projetos.

Ao professor Dr. Eduardo de Freitas Rocha Loures pela amizade e pelas conversas incentivadoras durante o tempo desta pesquisa.

Aos professores Dr. José Eduardo Pécora Júnior (UFPR) e Dr. Ângelo Márcio Oliveira Sant'Anna pelas opiniões, argumentações e contribuições extremamente relevantes para a conclusão deste trabalho.

Ao meu sobrinho, Professor Rodrigo Schaidt, pelas aulas de matemática que muito me ajudaram.

A minha esposa Celi e minhas filhas Bruna e Giulia, pelo apoio.

Aos demais familiares e amigos que de alguma forma ou outra me ajudaram.

Resumo

A variabilidade é um tipo de flexibilidade aplicada ao processo empresarial. Problemas de variabilidade para lidar com um mesmo processo para diferentes contextos. Cada um desses contextos diferentes é chamado de variante do processo. Por exemplo, o mesmo processo executado em diferentes países pode exigir variações na sequência de tarefas devido a diferenças nas regulamentações. Cada uma dessas variações para a seqüência de tarefas é uma variante do processo. Variantes de processo também podem surgir da variabilidade de produtos e serviços, diferentes grupos de clientes e diferenças temporais. Um conjunto de variantes diferentes de um mesmo processo é chamado de família de processos. Para as empresas, projetar e implementar cada variante do processo a partir do zero e mantê-la separada seria ineficaz e onerosa. Assim, há um grande interesse em capturar o conhecimento comum do processo apenas uma vez e reutilizá-lo em termos de modelos de processo de referência. Nos últimos anos, vários estudos foram feitos para lidar com famílias de processos. Esses estudos propuseram técnicas e métodos para selecionar variantes da família de processos. No entanto, essas técnicas e métodos têm sido fortemente direcionados aos processos pré-especificados. Embora já existam estudos que desenvolvam métodos e técnicas para fornecer seleção de variantes a processos pré-especificados, há uma falta de estudos que abordam as variantes de seleção para processos baseados em restrições. Os processos baseados em restrições tomaram atenção considerável nos últimos anos devido à maior flexibilidade que eles fornecem ao usuário, em oposição aos processos pré-especificados. Assim, esta pesquisa fornece um estudo que traz fundamentos, técnicas e métodos para criar uma estrutura conceitual (PAIS conceitual) para fornecer seleção de variantes para processos baseados em restrições, ou seja, combinar variabilidade com frouxidão.

Palavras-chave: variabilidade. variantes de processo. processo de negócio. processos pré-especificados. processos baseados em restrições.

Abstract

Variability is a kind of flexibility applied to business process. Variability concerns to handle a same process to different contexts. Each of these different contexts is named process variant. For example, the same process executed in different countries may require variations to sequence of tasks due to differences in regulations. Each of these variations to sequence of tasks is a process variant. Process variants can also emerge from variability of products and services, different groups of customers, and temporal differences. A set of different variants from a same process is called processes family. For companies, designing and implementing each process variant from scratch and maintaining it separately would be inefficient and costly. Thus, there is a great interest in capturing common process knowledge only once and re-using it in terms of reference process models. In recent years, several studies have been made to deal with process families. These studies have proposed techniques and methods to select variants from process family. Nevertheless, these techniques and methods have been strongly targeted to pre-specified processes. Although there are already studies developing methods and techniques to provide selection of variants to pre-specified processes, there is a lack of studies that address the selection variants to constraints based processes. Constraints based processes have taking considerable attention at last years due the greater flexibility that they provide to the user, in opposite to pre-specified processes. Thus, this research provides a study that brings fundamentals, techniques and methods in order to make a conceptual framework (conceptual PAIS) to provide selection of variants to constraints based processes, i.e. to combine variability with looseness.

Keywords: variability. process variants. business process. pre-specified processes. constraints based processes.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introduction

From the late 1970s to the early 1990s, the data had the main attention in information systems. Storing and retrieving information got the main attention. Designers of the information systems often concentrated on the data models. Database management systems were considered to be the main part of the information systems. During the 1990s, the process also had the attention of engineers and designers. The result was an increasing number of business processes managed by information systems. These information systems were named Process-Aware Information Systems (PAIS) (DUMAS; AALST; HOFSTEDE, 2005; AALST, 2013; LEITNER; RINDERLE-MA, 2014).

PAIS has evolved in several technical aspects. New techniques are always being developed in order to provide new services and tools. This is required since the world is constantly changing. From several technical aspects that can be enumerated to PAIS, our research is interested in two of them: perspectives to be model in PAIS (GRAMBOW; OBERHAUSER; REICHERT, 2017; STROPPI; CHIOTTI; VILLARREAL, 2015), and types of flexibility that PAIS can provide (ZUGAL et al., 2015; SLAATS et al., 2016; UNGER; LEOPOLD; MENDLING, 2015). In the following, each of these two aspects is briefly presented.

There are six perspectives to PAIS: function, behavior, information, organization, operation and time (REICHERT; WEBER, 2012a). Function perspective defines which tasks must compose the process (KHLIF et al., 2017). Behavior perspective defines which rules must to guide the sequence of tasks execution (ROSING; SCHERUHN; FALLON, 2015). Information perspective defines which information is required and what is its sequence among tasks (MEYER et al., 2013). Organizational perspective defines which human resources should perform the tasks (LEE; HWANG, 2016). Operational perspective defines what are the tools and technological requirements the tasks need to be performed (BRAUN et al., 2015). Time perspective defines temporal constraints that need to be obeyed during process execution (AREVALO et al., 2016).

Our research is interested in defining tasks to processes and which are the permitted sequences to these tasks. So our research is interested in function and behavior perspectives. Thus, from now, all the considerations in this work are related to function and behavior perspectives.

There are two types of processes to PAIS: pre-specified processes (YOUSFI; SAIDI; DEY, 2016; COSTA; TAMZALIT, 2017), and knowledge intensive processes (GOEDERTIER; VANTHIENEN; CARON, 2015; CICCIO; MARRELLA; RUSSO, 2015).

Pre-specified processes has rigid events sequences (COSTA; TAMZALIT, 2017). They offer few alternatives for the user to change events sequences (KOPP et al., 2015). Sequence of tasks are well defined, and the user should follow this sequence with few alternatives to change it. Pre-specified processes are also called highly structured processes. They are modeled by imperative languages. Example of imperative languages are BPMN (ALLWEYER, 2016) or Petri Nets (REISIG, 2013), among other.

Knowledge-intensive processes has greater flexibility to execute tasks than pre-specified processes (MUNDBROD; BEUTER; REICHERT, 2015). Unlike pre-specified processes, knowledge-intensive processes do not have rigid events sequences, the user can choose the tasks to be executed, based on knowledge and professional experience (MUNDBROD; REICHERT, 2014). There are two types of Knowledge-intensive processes: loosely specified processes (MAGGI, 2013; DEBOIS et al., 2016) and data-driven processes (KOUFI; MALAMATENIOU; VASSILACOPOULOS, 2015). Our work is interested only in loosely specified processes. Loosely specified processes are modeled by a set of tasks and a set of constraints (GIACOMO et al., 2015). Any sequence of events is permitted since it fulfills the constraints (CARVALHO et al., 2016). Loosely specified processes are also named declarative processes or constraints based processes. From now, this work uses the term *constraints based processes* to refer to loosely specified processes. Constraints based processes are modeled by declarative languages. Example of declarative languages are Declare (MONTALI et al., 2013), Dynamic Condition Response (DCR) Graphs (SLAATS et al., 2013), and Supervisory Control Theory (SCT) Approach (SANTOS et al., 2014).

There are four types of flexibility to PAIS: looseness, variability, adaptation, evolution and looseness (REICHERT; WEBER, 2012a; AYORA et al., 2015). This work is interested in looseness and variability.

Looseness is related to knowledge-intensive processes (MARTIN, 2016). There are not events sequences previously defined in knowledge-intensive processes. Events sequences are defined by the user at same time the process is executed. This provides flexibility to the user to execute the tasks in accord to knowledge and professional experience. Since knowledge-intensive processes do not have rigid events sequences, they have loose specification. So knowledge-intensive processes provides Looseness. In other words, Looseness is the type of flexibility that provides great power to the user to choose the tasks to be executed in a process. This power is supported by the user's knowledge and professional experience. This is in opposition to pre-specified processes.

Variability concerns to handle a same process in different contexts (ROSA et al., 2017). Each of these different contexts is named process variant or just variant (VALENÇA et al., 2013). For example, the same process executed in different countries may require different tasks sequences due to differences in regulations. Each of these tasks sequences is a process variant. Process variants can also arise from variability of products and services, different groups of customers, and temporal differences (AYORA et al., 2015).

A set of variants from a same process is called processes family (GRÖNER et al., 2013).

So, variability is the type of flexibility that provides power to the user to select a process variant from a process family .

Designing and implementing each process variant from scratch and maintaining it separately is costly (LEE; HWANG, 2016). There is a great interest in capturing common process knowledge only once and re-using it in terms of reference process models (AYORA et al., 2016). Examples of reference process models are Information Technology Infrastructure Library (ITIL) in Information Technology (IT) Service Management (IDEN; EIKEBROKK, 2013; MARRONE et al., 2014; CARDOSO, 2015), Reference Processes in SAP's ERP System (LEON, 2014), or Medical Guidelines (BARR et al., 2013; KATZNELSON et al., 2014; CHOU et al., 2016; LYMAN et al., 2015). Although reference process models foster the reuse of common process knowledge, they usually lack comprehensive support for explicitly describing variations (REICHERT; WEBER, 2012a; AYORA et al., 2015).

In recent years, several studies have addressed topics related to processes families (AYORA et al., 2013b; NATSCHLÄGER et al., 2016). These studies have proposed techniques and methods to select variants from process families (ROSA et al., 2013). These methods are usually called *methods to selection of variants*. Nevertheless, these techniques and methods have been strongly targeted to pre-specified processes (ROSA et al., 2017). This happens because researchers usually want to identify pre-specified processes as from reference process models (AYORA et al., 2016; REICHERT; HALLERBACH; BAUER, 2015). Implementing a lot of tasks sequences in a pre-specified process can demand a lot of gates (YOUSFI; SAIDI; DEY, 2016). This can be costly because makes the process modeling hard and prone to errors. Nevertheless, when the user applies some method to select variants from pre-specified processes, it is easier defining a set of variants (a processes family) that complies with a set of different contexts. This happens because the methods of selection of variants provide techniques that allow the user to act properly at three different times: design time, configuration time and run time (AYORA et al., 2015).

At design time, the methods to selection of variants makes the modeler able to model each variant of the pre-specified process and mix all of them into the same model. This model is usually named configurable process model. At design time, the modeler must define the method to support the user to select the variants at configuration time. There are, at least, four methods to support the user to select the variants at configuration time: questionnaires, features, goals models, decisions tables (AYORA et al., 2015). This work is interested only in questionnaires. Thus, modeler defines the configurable process model with the set of variants (the processes family) of the pre-specified process, and the questionnaire to support the user to select the variants from the configurable process model.

At configuration time, the methods to selection of variants makes the user able to answer the questionnaire. User answer the questionnaire and select a process variant from the configurable process model. At run time, the process variant selected from the configurable process model is executed.

Although researchers usually define pre-specified processes as from reference process models, some reference process models are modeled in a more suitable way by constraint based processes (ROVANI et al., 2015). There are already studies developing methods and techniques to provide selection of variants to pre-specified processes. But there is a lack of studies addressing selection of variants to constraints based processes.

Constraints based processes have taken considerable attention at last years because they provide greater flexibility than pre-specified processes (REIJERS; SLAATS; STAHL, 2013; MERTENS; GAILLY; POELS, 2015b; UNGER; LEOPOLD; MENDLING, 2015). As previously mentioned, there are studies that provide frameworks and approaches for modeling constraint based processes. These studies propose methods and techniques to provide only looseness to the processes, but they do not provide variability. But, we understand that providing variability to constraint based processes, i.e. combining variability with looseness, can bring several advantages. We believe that the user can take these advantages at design time, configuration time and run time in PAIS.

Thus, this research provides a study that brings fundamentals, techniques and methods to make a conceptual framework (conceptual PAIS) to select variants to constraints based processes, i.e. to combine variability with looseness. In the following subsection, we present the advantages we believe that can be taken at design time, configuration time and run time in PAIS. These advantages are our justification to address this topic.

1.1 Justification

This section presents the justification to our research.

The first justification is related to define a constraint based process as from a reference process model. As mentioned previously, although, researchers usually define pre-specified processes as from reference process models, some reference process models are better supported through constraint based processes. This can be demonstrated by examples of constraints based processes. Among these examples, there are constraints based processes to model healthcare system (TELANG; KALIA; SINGH, 2015). But a constraints based process to healthcare system could be derived from a reference process model like Medical Guideline (MERTENS; GAILLY; POELS, 2015a). So from the same Medical Guideline, we can define other constraints based processes intended to healthcare system. Thus, this set of constraints based processes would define a processes family to a healthcare system based on a Medical Guideline. Other example is demonstrated in this research. In this research, we use Project Management Body of Knowledge (PMBOK) (SNYDER, 2014) to derive a set of process variants modeled by a constraints based process. PMBOK is a reference process model to project management. These two examples demonstrate that process variants can also be modeled by constraints based processes.

The second justification is related to the increasing interested in constraint-based processes in recent years. Constraints-based processes have received increased interest because they provide a non-standardized setting (REIJERS; SLAATS; STAHL, 2013). Constraints-based processes are modeled by a set of tasks and a set of constraints. Users are able to easily identify what are the constraints to be obeyed by the process (GOEDERTIER; VANTHIENEN; CARON, 2015). If compared with pre-specified processes, this can be an advantage because the user do not need to describe the whole process with all the tasks sequences. Users just have to specify the tasks and the constraints of the process. Non-standardized setting provides power to the user to choose tasks to be executed in a process (MERTENS; GAILLY; POELS, 2015b). In general, the user makes these choices in accord to professional expertise and the context in which the process is performed (UNGER; LEOPOLD; MENDLING, 2015). If compared with pre-specified processes, this requires a greater expertise from the user. But at same time, it can be also a great advantage to the user. Justifications 1 and 2 are enough to justify a study about selection to variants as from constraints based processes. Nevertheless, we present two more justifications.

The third justification concerns to the user support that selection of variants (variability) can provide to constraints based processes (looseness). First, we should analyze design time, configuration time and run time. We suppose a constraint based process modeled to comply with a great number of process variants. This process does not have variants selection support support. So the constraint based process probably be modeled with a lot of tasks and constraints. At run time, the user has to choose one out of a lot of sequences of events that can be followed, to comply the process objectives. But, each sequence of events is related to a specific process context (process variant). So the user must have, at run time, a wide knowledge of all the process variants. This is required because the user must be able to identify each process application context. After identifying the process application context, the user must have expertise to identify the tasks to be executed inside this context. This condition can be complex to the user since there are a lot of application's contexts, and each of them has a lot of tasks. Now, we suppose a constraint based process modeled to comply with a great number of process variants. This constraint based process is modeled with a lot of tasks and constraints. But, this process has variants selection support. So before run time, there is the configuration time. At configuration time, the user has the support of a questionnaire to select process variants. Questionnaire provides a finite set of features to be selected. So the user focus the professional experience to this finite set of features. At run time, the user must select the tasks to be executed in only one application's context. At configuration time, questionnaire brings ease to the user to choose the application's context. Thus, methods to select variants (variability) improve user support to constraints based process (looseness).

The fourth justification concerns to improve the power of the user to specify tasks and constraints. Firstly, we suppose a constraint based process modeled to comply with two variants: *variant 1* and *variant 2*. In *variant 1*, there is a constraint that imposes that *task 1* must be executed before *task 2*. In *variant 2*, there is a constraint that imposes that *task 2* must be executed before *task 1*. These constraints are conflicting. This conflict impedes that *task 1* and *task 2* be executed. There is not variants selection support, i.e. there is not configuration time. So at run time, the user cannot complete any task because there is a conflict between process constraints. We can suppose other case. There is a process modeled to comply with two variants: *variant 3* and *variant 4*. In *variant 3*, there is a constraint that imposes that whenever *task 3* is executed, *task 4* must be executed afterward. In *variant 4*, there is a constraint that imposes that whenever *task 4* is executed, *task 3* must be executed afterward. These constraints are conflicting. This conflict impedes that the process be finished. There is not variants selection support, i.e. there is not configuration time. So at run time, user cannot finish the process. These two cases demonstrate that if process variants are inserted into the same constraint based process, with no variants selection support, then problems related to tasks execution can arise. In these cases, if variants selection support is applied then these problems can be solve. If variants selection support is applied, then there is configuration time. At configuration time, user selects exactly one variant. So at run time, there is only one variant to be executed. There is not mix of conflicting constraints. Thus, variants selection methods (variability) improve the power of the user to specify tasks and constraints to constraints based process (looseness).

1.2 Research question

In section *Introduction* we demonstrated that the focus of our research is the application of variants selection methods to constraints based processes. In section *Justification*, we argued that there are at least four reasons to justify that our research is important. In this section we are going to present variants selection's fundamentals and our research question.

Literature provides several topics to be complied to implement a framework to variants selection's support (AYORA et al., 2015). We selected five of them to define the scope of our research. They are: (i) definition of a language to model process variants, (ii) syntactic and semantics correctness of process variants, (iii) map as from process variants to domain facts, (iv) logical and temporal consistency between domain facts, (v) grouping domain facts into questions. We believe that if we concentrate on these five topis, we are able to propose a framework to select variants from constraints based processes. Our research question is related to these five topics and how they can be dealt to make a framework (PAIS) to select variants from constraints based processes. Next, we present a brief description of each of these five topics.

Variants selection fundamentals provide that it is required to define the variants modeling language. For pre-specified processes, there are already languages to this purpose. Configurable Event-driven Process Chains (C-EPC) (RIEHLE et al., 2016), Process Variants by Options (Provop) (SARNO et al., 2015), Process Family Engineering in Service Oriented Applications (PESOA) (WESKE, 2006) are frameworks to variants selection to pre-specified processes. C-EPC framework provides C-EPC language. Provop framework provides a language-independent approach. PESOA framework provides a set of techniques that may be applied to any imperative language. As far as we know, for constraints based processes, , there are only frameworks for processes modeling, but with no variants selection support. For example, Declare/LTL (PESIC, 2008; MONTALI et al., 2013; CICCIO et al., 2015), DCR Graphs (MUKKAMALA, 2012) and SCT approach (SANTOS et al., 2014) provide process modeling support, but none of them provides any variants selection support. If we chose one of these approaches for modeling processes variants, we would have to provide it with external elements to enable variants selection support. This can brings advantages since we would not need to worry about developing a declarative language from scratch. But, at same time, we would have to make connections and adaptions to enable variants selection support to the chosen approach. For Declare/LTL and DCR Graphs frameworks, these connections and adaptions probably would be hard to do, since they are ready frameworks. SCT approach encompasses a lot of constructs, and provides support to other elements which are out of the scope of our research, such as uncontrollable events (RAMADGE; WONHAM, 1987). That can bring difficulties to implement variants selection support. On the other hand, defining a new declarative language from scratch probably would bring a lot of work at first. However, that could ease to develop a language in accord to the fundamentals of variants selection. No adaptations would be required, the language is created from scratch to fulfill variants selection's fundamentals.

Variants selection fundamentals provide that it is required to ensure process variants correctness (ROSA, 2009). Process variants correctness is related to the syntax correctness and semantics correctness from the modeling language. For pre-specified processes, there are already models that address that topic. That is the case of frameworks C-EPC, Provop and PESOA. Frameworks C-EPC, Provop and PESOA offer technical resources to allow the user to model each process variants in accord to language's syntactic and semantics rules. At design time, process variants are mixed at the same process by techniques to preserve their syntactic and semantics features. These process are called configurable process model. At configuration time, techniques are used to select process variants and to keep them syntactic and semantics preserved. At run time, the preserved syntactic and semantics features ensure the process variant be performed properly. With regard to constraints based processes, Declare/LTL and DCR Graphs frameworks provide support to syntactic and semantics correctness. But, they do not provide any variants selection support. SCT approach still does not provide any syntax and semantics rules. Making a new declarative language from scratch would permit to precisely define language's syntax and semantics rules. This could be much advantageous since the syntax and semantics rules, and the techniques to deal with them, would be made from scratch, in other words, no adaptation would be required.

Variants selection fundamentals provide that pre-specified processes must be composed by a set of fixed tasks and a set of variable tasks (REICHERT; WEBER, 2012a). Fixed tasks are always executed in every application context. Variable tasks are not executed in every application context. Each variable task is executed only in some application contexts. The process regions with variable tasks are called variation points. Selection of the application context and variable tasks are implemented by domain facts and process facts. Domain fact is a process feature which can vary in accord to the application context. Process fact is a set of variable tasks which is performed if a set of domain facts is selected. In fact, a process fact is an option from a variation point. So when the user selects a set of features (domain facts), set of tasks (process facts) are selected. For example, after the initial exam, a physician is able to define a set of features (domain facts) to the patient. In accord to the set of features (domain facts), a set of tasks for the patient treatment (a set of process facts from variation points) is selected. Mapping domain facts to process variants are usually implemented by logic sentences. For example, a mapping as from features to a process variant could be expressed by a sentence like *if feature 1 and feature 2 and feature 3 are true, or if feature 4 and feature 5 and feature 6 are true, then the process variant 1 is selected*. Although there are studies addressing domain facts, process facts, and variation points to pre-specified processes, there is lack of studies addressing these topics to constraints based processes.

Variants selection fundamentals provide that logical and temporal consistency must be guaranteed to domain facts (AYORA et al., 2015). There are already studies addressing this topic to pre-specified processes. As from these studies we enumerate four logical/temporal relations to be imposed to domain facts: mutual exclusion, mutual inclusion, implication, and precedence. Mutual exclusion defines that if a domain fact is selected then the other cannot be selected and vice verse. Mutual inclusion defines that if a domain fact is selected then the other must be selected and vice verse. Implication defines that if a domain fact is selected then the other must be selected. Precedence defines that a domain fact must be set out before the other during configuration time. All the domain facts relations must be consistent to each other. Inconsistencies happen when two relations impose contradictory behavior among domain facts. For example, the specifications *if domain fact 1 is selected then domain fact 2 must be selected and vice verse*, and *if domain fact 1 is selected then domain fact 2 must not be selected and vice verse* are inconsistent. Domain facts relations also must be checked to identify tasks that cannot be executed. For example, the specifications *task 1 must precede task 2* and *task 1 excludes task 2* set out that *task 2* will never be executed. Although the existent studies are intended to pre-specified processes, they can support our research. This happens because logical and temporal consistency between domain facts do not depend on the type of the process. That is exclusively a logical issue.

Variants selection fundamentals provide that domain facts must be grouped into questions. There are already studies proposing some formalization to questionnaires and questions (ROSA, 2009). In general, these studies define questionnaires as a set of questions, and define questions as a set of domain facts. Functions map questions to domain facts. Question provide sets of domain facts. Questions inherit logical and temporal consistency from domain facts. For example, the specification *if domain fact 3 is selected then domain fact 4 is selected*, sets that whenever *domain fact 3* is selected, *domain fact 4* is selected even the question with *domain fact 4* was not answered yet. An example of inheritance as from temporal consistency is presented next. The specification *domain fact 1 must be set before domain fact 2*, sets that the question with *domain fact 1* must be answered before the question with *domain fact 2*. Although the studies are intended for pre-specified processes, they can support our research. This happens because formalization to questionnaires, questions and domain facts do not depend on the type of the process.

After presenting a brief description of the previous five topics, we set our research question. This is done next.

Given that: (i) definition of a process variants' language, (ii) syntactic and semantics correctness of process variants, (iii) map as from process variants to domain facts, (iv) logical and temporal consistency for domain facts, (v) grouping of domain facts into questions, are the issues to be complied with, the following research question is proposed: *how may these five issues be dealt by a set of methods and techniques, to make a framework to select variants from constraints based processes?*

1.3 Objectives

This section presents the main and the specific objectives of our research. The main objective is derived from research question set in the last section. To be able to fulfill the main objective, five specific objectives are derived. By fulfilling these five objectives, the main objective is fulfilled. The main and specific objectives are presented next.

Main objective (MO). Propose a conceptual framework to select variants as from constraints based processes. The framework that we propose in this research is supported by fundamentals from Process Aware Information Systems (PAIS), selection of variants and constraints based processes. The framework selects variants from only a type of the process: constraints based processes. The framework covers two perspectives: function and behavior. It covers Function perspective because a constraints based process defines a set of tasks. It covers Behavior perspective because a constraints based process defines a set of constraints to restrains tasks events sequences. The framework covers two types of flexibility: looseness and variability. It covers looseness because it models, configures and runs constraints based processes. It covers variability because it provides variants selection for constraints based processes. The framework covers three times: design time, configuration time and run time. It covers design time because it provides a sub-framework to design all the process variants, and mix all of them into the same configurable process model. It covers configuration time because it provides a sub-framework for variants selection. It covers run time because it provides a sub-framework to run the process variant that was selected at configuration time.

Specific Objective 1 (*SO1*): *Define a constraints based language to model process variants.* There are at least four options of constraints based language. The first one is Declare/LTL framework. Declare is based on Linear Temporal Logic (LTL). Declare provides a set of graphics constructs to model, verify, and run constraints based processes. Declare offers a great number of constraints (more than twenty), and provides interface to make new constraints. The second one is DCR Graphs framework. DCR Graphs is based on set's operations. DCR Graphs also provides a set of graphics constructs to model, verify, and run constraints based processes. DCR Graphs offers five constraints, and it does not provide interface to make new ones. The third one is SCT approach. SCT approach is a mathematical formalism based on Supervisory Control Theory. SCT approach does not provide any set of graphic support to model, verify, and run constraints based processes. SCT approach also offers a great number of constraints (more than twenty), and it is able to implement new constraints. The fourth one is setting a new language. As previously mentioned, defining a new declarative language demands additional effort, but it can be advantageous because we can make it from scratch in accord to the necessary requirements. These requirements, of sure, would be in accord to the fundamentals of variants selection.

Specific Objective 2 (*SO2*): *Propose a framework for design time.* Framework for design time must provide support for user to model process variants. Every process variant must comply with the language's syntax and semantics defined in *SO1*. Framework for design time must provide support for user to mix all the process variants at the configurable process model. Configurable process model must be also made in accord to the language syntax and semantics. Framework for design time must provide support for user to define process fact. In pre-specified processes, a set of process facts represents a set of variable tasks. Framework for design time must provide support for user to make logic relations between domain facts and process facts. These relations must preserve logic and temporal consistency and, at the same time, all process variants must be able to be selected. Framework for design time must provide support for user to group domain facts into questions. Temporal and logical constraints applied to domain facts are inherited by the questions.

Specific Objective 3 (*SO3*): *Propose a framework for configuration time.* Framework for configuration time must provide support to user to select domain facts (features) from processes. This must be done by some user interface. That user interface could be provided by a more refined graphical interface or just by a text interface. The user interface must be able to take the domain facts selected the user. Framework for configuration time must provide support for simplifying and reducing the logic sentences. Whenever the user sets a domain fact (as TRUE or FALSE), domain facts' logic sentences need to be simplified. This simplification reduces the amount of variables in the logic sentences. Framework for configuration time must provide support for presenting the questions to the user. This must be in accord to the domain facts' logic and precedence rules. Domain facts' logical and precedence rules are inherited by questions. Thus some method or technique must be suggested to address this issue. Framework for configuration time must provide support for user to identify which are the selected (TRUE) and not selected (FALSE) process facts. Some approach to address that topic must be presented.

Specific Objective 4 (*SO4*): *Propose a framework for run time.* Framework for run time must provide support to user to know which are the enabled tasks events at each process step. Each approach uses some technique to calculate the enabled tasks events at each process step. For example, Declare/LTL uses Linear Temporal Logic, DCR Graphs uses sets' operations, and SCT approach uses method synchronous product, method to exclude bad states, and method to exclude blocking states. We can use some of these techniques or propose a new one. Framework for run time must provide support for user to know which are the pendent tasks to be executed at each process step. Declare/LTL does not inform which are pendent tasks, but it informs which are the not fulfilled constraints. User must perform events sequence to fulfill the not fulfilled constraints. DCR Graphs informs which are the events that are pendent. User must perform events sequence to execute the pendent events. SCT approach does not provide any technique to inform which are the enabled events. We are going to propose some technique to inform which are the enabled and pendent events in our framework.

Specific Objective 5 (*SO5*): *Demonstrate the application of the framework.* Reference process models captures common process knowledge only once to re-using them repeated times. It captures common knowledge as from several process application contexts. Each process application context corresponds to a process variant. Techniques and methods have been targeted to derive process variants as from pre-specified processes. Nonetheless, some cases of application of reference process models can be better modeled by constraint based processes. So we want to do two demonstrations through *SO5*. The first one is model the variants as from an actual reference process model by using a constraints based language. The second one is demonstrate the virtual operation of our framework presenting the steps at design time, configuration time, and run time. Information Technology Infrastructure Library (ITIL) in Information Technology (IT) service management, reference processes in

SAP's ERP system, medical guidelines, and Process Management Body Of Knowledge (PMBOK) are examples of reference process models. Any of them can be suitable to be used as the reference process model to fulfill *SO5*. Other reference process models can be researched in order to be used to that objective.

1.4 Research method

This section presents the method (process) used to develop our research. This research process is composed by a sequence of tasks. These tasks generate data to fulfill the five *Specific Objectives* previously defined in section 1.3. If all the *Specific Objectives* are fulfilled then the *Main Objective* is also fulfilled. Figure 1 shows the research method. Research process is divided in three phases. These phases are explained next.

1.4.1 Phase 1

Phase 1 of the research process encompasses three tasks: *Do literature review to reference process models* (t_1), *Do literature review to selection of variants* (t_2), *Do literature review to declarative languages* (t_3). Tasks sequence execution at *Phase 1* is $t_1.t_2.t_3$. From these three tasks are generated three data: *Literature review to reference process models* (d_1), *Literature review to selection of variants* (d_2), and *Literature review to declarative languages* (d_3). d_1, d_2 and d_3 cover all the literature review required to develop our research and fulfill the five *Specific Objectives*. Tasks and data at *Phase 1* are described next.

Task *Do literature review to reference process models* (t_1) has no input. We execute t_1 to identify and organize reference process models' fundamentals. t_1's output is *Literature review to reference process models* (d_1). d_1 must cover the following topics: history, features, and description of reference process models. t_1 is the first task to be executed at *Phase 1* because it is able to provide a reference process models broader view. So we think that t_1 can bring great contribution for next research steps.

Task *Do literature review to selection of variants* (t_2) has an input: *Literature review to reference process models* (d_1). t_2's output is *Literature review to selection of variants* (d_2). d_2 must cover the following topics: logical/temporal relations of domain facts and process facts, questionnaire, variants modeling, configurable process models, variation points, syntactic and semantics correctness.

Task *Do literature review to declarative languages* (t_3) has an input: *Literature review to selection of variants* (d_2). t_3's output is *Literature review to declarative languages* (d_3). t_3 uses d_2 to address the important features for variants selection.

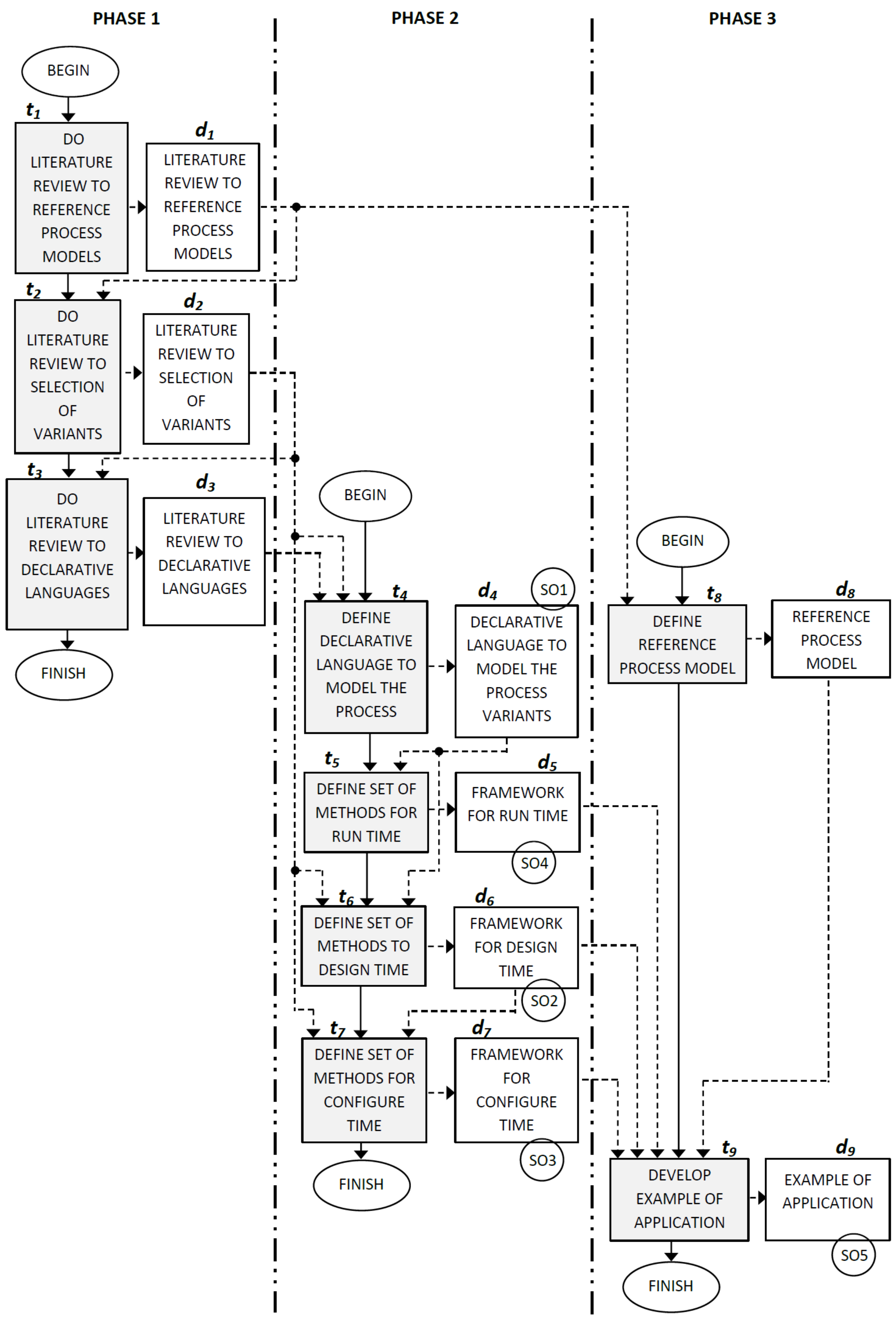

PHASE 1
PHASE 2
PHASE 3
BEGIN
t_1
DO LITERATURE REVIEW TO REFERENCE PROCESS MODELS
d_1
LITERATURE REVIEW TO REFERENCE PROCESS MODELS
t_2
DO LITERATURE REVIEW TO SELECTION OF VARIANTS
d_2
LITERATURE REVIEW TO SELECTION OF VARIANTS
t_3
DO LITERATURE REVIEW TO DECLARATIVE LANGUAGES
d_3
LITERATURE REVIEW TO DECLARATIVE LANGUAGES
FINISH
BEGIN
t_4
DEFINE DECLARATIVE LANGUAGE TO MODEL THE PROCESS
d_4
SO1
DECLARATIVE LANGUAGE TO MODEL THE PROCESS VARIANTS
t_5
DEFINE SET OF METHODS FOR RUN TIME
d_5
FRAMEWORK FOR RUN TIME
SO4
t_6
DEFINE SET OF METHODS TO DESIGN TIME
d_6
FRAMEWORK FOR DESIGN TIME
SO2
t_7
DEFINE SET OF METHODS FOR CONFIGURE TIME
d_7
FRAMEWORK FOR CONFIGURE TIME
SO3
FINISH
BEGIN
t_8
DEFINE REFERENCE PROCESS MODEL
d_8
REFERENCE PROCESS MODEL
t_9
DEVELOP EXAMPLE OF APPLICATION
d_9
EXAMPLE OF APPLICATION
SO5
FINISH

Figura 1 – Research method's tasks and data (research process)

1.4.2 Phase 2

Phase 2 of the research process encompasses four tasks: *Define declarative language to model the process variants* (t_4), *Define set of methods to run time* (t_5), *Define set of methods to design time* (t_6), *Define set of methods to configuration time* (t_7). The execution sequence of tasks at *Phase 2* is $t_4.t_5.t_6.t_7$. From these four tasks are generated four data: *Declarative language to model the processes variants* (d_4), *Framework for run time* (d_5), *Framework for design time* (d_6), *Framework for configuration time* (d_7). Tasks and data in *Phase 2* are described next.

Task *Define declarative language to model the process variants* (t_4) has two inputs: *Literature review to selection of variants* (d_2) and *Literature review to declarative languages* (d_3). t_4's output is *Declarative language to model the processes variants* (d_4). t_4 is the first task to be executed at *Phase 2*. This is important because all the operations at design time, configuration time and run time are defined as from process variants language (d_4). d_4 fulfills *Specific Objective 1* (*SO1*).

Task *Define set of methods to run time* (t_5) has an input: *Declarative language to model the process variants* (d_4). t_5's output is *Framework for run time* (d_5). t_5 is the first task executed after t_4 because the methods to run the process variant can be defined independently. Any method to run the process variant process is related only to modeling language, so it does not depend on other methods at design time and configuration time. d_5 fulfills *Specific Objective 4* (*SO4*).

Task *Define set of methods to design time* (t_6) has two inputs: *Literature review to selection of variants* (d_2) and *Declarative language to model the processes variants* (d_4). t_6's output is *Framework for design time*. d_6 fulfills *Specific Objective 2* (*SO2*).

Task *Define set of methods to configuration time* (t_7) has two inputs: *Literature review to selection of variants* (d_2) and *Framework for design time* (d_6). t_7's output is *Framework for configuration time* (d_7). d_7 fulfills *Specific Objective 3* (*SO3*).

1.4.3 Phase 3

Phase 3 of the research process encompasses two tasks: *Define reference process model* (t_8) and *Develop example of application* (t_9). The tasks execution sequence at *Phase 3* is $t_8.t_9$. From these two tasks are generated two data: *Reference process model* (d_8) and *Example of application* (d_9). Tasks and data in *Phase 3* are described next.

Task *Define reference process model* (t_8) has an input: *Literature review to reference process models* (d_1). We execute t_8 to analyze the features of the reference process models covered by literature review. t_8's output is *Reference process model* to be used in *Example of application* (d_9).

Task *Develop example of application* (t_9) has four inputs: *Framework for design time* (d_6), *Framework for configuration time* (d_7), *Framework for run time* (d_5), and *Reference process model* (d_8). We execute t_9 to present *Example of application. Example of application* must cover modeling, configuration and run of process variants. *Example of application* fulfills *Specific Objective 5* (*SO5*)

1.4.4 Expected results

Expected results are directly associated to the five *Specific Objectives*. In other words, the expected result is fulfill the five *Specific Objectives* and consequently, the *Main Objective*. Expected results in each *Specific Objective* are listed next.

For *Specific Objective 1* (*Define a constraints based language to model the process variants*), the expected results are:

- $ER_{1.1}$: Tasks and constraints must be represented by well defined mathematical models.
- $ER_{1.2}$: Accurate rules to constraint based language's syntax and semantics: these rules must describe precisely how to combine constraint based language's constructs.

For *Specific Objective 2* (*Propose a framework for design time*), the expected results are:

- $ER_{2.1}$: Make process variants: framework must provide some procedure to make each process variant in accord to syntactic and semantics rules of the constraints based language.
- $ER_{2.2}$: Mix process variants: framework must provide some procedure to mix all the process variants into the same constraints based process.
- $ER_{2.3}$: Questionnaire support: framework must provide some procedure to make the questionnaire. Questionnaire supports the user to select process variants at configuration time.

For *Specific Objective 3* (*Propose a framework for configuration time*), the expected results are:

- $ER_{3.1}$: Questionnaire support: At configuration time, framework must provide some procedure to support the user to answer the questionnaire.
- $ER_{3.2}$: Support to select process variants: At configuration time, framework must provide some procedure to identify and select a process variant in accord to syntax and semantics rules of the constraints based language.

For *Specific Objective 4* (*Propose a framework for run time*), the expected results are:

- $ER_{4.1}$: User Support to inform which are the tasks that must be executed. At run time, framework must provide some procedure to identify which are the tasks required to be executed at each process step.
- $ER_{4.2}$: User Support to inform which are the tasks that can be executed. At run time, framework must provide some procedure to identify which are the tasks that can be executed at each process step.

For *Specific Objective 5* (*Demonstrate the application of the framework*), the expected results are:

- $ER_{5.1}$: Present a reference process model with features and domain application.
- $ER_{5.2}$: Define a set of application contexts for reference process model.
- $ER_{5.3}$: Demonstrate the set of application contexts at design time, configuration time and run time.

1.5 Articles in this research

This section describes shortly each article of this document.

Article 1 is *Modeling Constraint-based Processes: a Supervisory Control Theory Application.* This article presents a study about constraints based process and propose an approach to model constraints based process by using methods of Supervisory Control Theory (SCT). It presents a template to model tasks and a set of templates to model constraints. Each process is composed by a set of tasks and a set of constraints. *Article 1* uses constraints based process' fundamentals to demonstrate its application in variants selection. *Article 1* is related to *SO1.*

Article 2 is *Selection of process variants from pre-specified processes based on supervisory control theory.* This article presents a study about selection of variants for pre-specified processes. It proposes to apply constraints in some points of pre-specified processes in order to model process variants. Modeling of these constraints is ruled by SCT formalism. *Article 2* also uses constraints based processes' fundamentals to demonstrate its application in variants selection. It is the continuity of *Article 1.* textitArticle 2 is also related to *SO1.*

Article 3 is *Simple Declarative Language (SDL): a conceptual framework to model constraint based processes. Article 3* defines Simple Declarative Language (SDL). SDL is a conceptual framework to model constraints based processes. This paper defines the syntactic and semantically rules to the processes modeled by SDL framework. frameworks for design and run time are described from syntactic and semantically rules. *Article 3* finish the study about constraints based process to model process variants. *Article 3* also proposes a framework to run the processes modeled by SDL. *Article 3* is related to *SO1* and *SO4*.

Article 4 is *A conceptual framework to select variants from constraint-based processes*. *Article 4* defines Selection of Variants with Simple Declarative Language (SVSDL). SVSDL is a conceptual framework to provide variants selection support to SDL processes. SVSDL is divided into three frameworks: framework for design time, framework for configure time and framework for run time. Since framework for run time is the same of *Article 3*, *Article 4* is related to *SO2*, *SO3*, and *SO4*.

Article 5 is *An approach for selection process variants from PMBOK*. *Article 5* presents some processes management models, including Process Management Body Of Knowledge (PMBOK). This paper demonstrates an application where SVSDL is used to select variants from PMBOK processes. *Article 5* is related to *SO5*.

1.6 Document structure

This document is divided in 7 sections. *Section 1* presents the context, justification, research question, research objectives, research method, expected results, articles summary, and document structure. *Section 2* presents *Article 1*. *Section 3* presents *Article 2*. *Section 4* presents *Article 3*. *Section 5* presents *Article 4*. *Section 6* presents *Article 5*. *Section Conclusions* presents the final evaluation of the main and specific objectives. That section analyzes which objectives are complied with and which techniques are utilized to do that.

2 Modeling Constraint-based Processes: a Supervisory Control Theory Application

Abstract

Constraint-based processes require a set of rules that limit their behavior to certain boundaries. In these processes, the control flow is defined implicitly as a set of constraints or rules, and all possibilities that do not violate any of the given constraints are allowed to be executed. The present paper proposes a new approach to deal with constraint-based processes. The proposed approach is based on Supervisory Control Theory, a formal foundation for building controllers for discrete-event systems. The controller proposed in this paper monitors and restricts execution sequences of activities such that constraints are always obeyed. We demonstrate that our approach may be used as a declarative language for constraint-based processes. In order to provide support for users of such processes and to facilitate the using of our control approach, we offer a set of constraints modeled by automata. This set encompasses the constraints frequently needed in workflow system.

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

2.1 Introduction

Nowadays constraint-based processes approaches have received increased interest (REICHERT; WEBER, 2012a). In these processes, the control flow is defined implicitly as a set of constraints or rules, and all possibilities that do not violate any of the given constraints are allowed to be executed (PESIC; SCHONENBERG; AALST, 2007) (HILDEBRANDT; MUKKAMALA; SLAATS, 2012) (HILDEBRANDT; MUKKAMALA; SLAATS, 2011). A constraint-based process model specifies the activities that must be performed to produce the expected results but it does not define exactly how these activities should be performed (FAHLAND et al., 2009b) (FAHLAND et al., 2010). Thus, any execution order of activities is possible provided that the constraints are not violated. Thus, most of time the process execution is driven by users choice.

DECLARE (PESIC; SCHONENBERG; AALST, 2007) (AALST; PESIC; SCHONENBERG, 2009) is developed as a constraint-based system and it uses a declarative language grounded in Linear Temporal Logic (LTL). DECLARE provides a graphical representation of constraints (DecSerFlow) (PESIC; AALST, 2006) that hides the associated LTL formulas from users. According to (HILDEBRANDT; MUKKAMALA; SLAATS, 2012)

(HILDEBRANDT; MUKKAMALA; SLAATS, 2011), this approach suffers from the fact that the subsequent tools for execution and analysis will refer to the LTL expression and not to the graphical notation. The full generality of LTL may lead to a poor execution time. For verification and enactment purposes, it is necessary to translate LTL to finite automata. While computers are very good at handling nite automata, the translation itself is often a roadblock as it may take time exponential in the size of the LTL formulas (WESTERGAARD, 2011). This motivates researching the problem of finding an expressive constraint-based processes approach where both the constraints as well as the run time state can be easily visualized and understood by the end user and also allows an effective verification (blocking,conflict, dead tasks) and execution of activities.

In the present paper we propose a new approach to deal with constraint-based processes founded on the Supervisory Control Theory (RAMADGE; WONHAM, 1989). The new approach proposes a control system which restrains the process in order to not violate the constraints. This action is accomplished through dynamic disabling of some events, restraining the state space of process. We consider that a process may contain sequences of events that are not allowed to occur. These sequences may violate a desired ordering of events and they need to be avoided. Thus, a supervisor is built in order to ensure that the whole set of constraints is not violated. We had some challenges bringing the formal foundation of SCT into a constraint-based process model, which characterizes the originality of our paper. We highlight the following contributions:

1. A new approach to deal with constraint-based processes. The proposed approach is based on SCT. The supervisor obtained applying SCT monitors and restricts execution of sequences of activities such that constraints are always obeyed. We demonstrate that our proposal can be used as a declarative language for constraint-based processes. Our approach does not limit the user by imposing rigid control-flow structures. In fact, the basis of our approach is to inform users of which activities are not allowed to occur after an observed trace of events at run-time, and users operate with some freedom because they choose execution sequences allowed under supervision;

2. A new approach to audit processes. Applying SCT results in a language (sequence of events) that considers all possible sequences which do not violate any of the constraints imposed to the process. It is possible to audit an execution of a process comparing if the performed sequence of activities belongs to that language.

3. Modeling activities and constraints using automata. We represent activities and constraints frequently needed in workflow systems using automata. This is necessary to apply the SCT. We propose a general model of activities as well as a set of

constraints (Fig. 9 to 12, section 2.4). We aim to support users without a deep knowledge in SCT on its application in order to model constraint-based processes;

The present paper is an extended version of the previous paper presented in WorldCist 2013 (SCHAIDT et al., 2013), and is organized as follows: Section 2.2 describes the Supervisory Control Theory, as the fundamental concept of the proposed approach. Section 2.3 explains the modeling of activities using automata. Section 2.4 explains the modeling of constraints using automata. Also, it is presented four categories of constraints usually needed in business processes. Section 2.5 presents an application example to illustrate our approach. Section 2.6 discusses the process execution and the architecture of the run-time environment. Section 2.7 concludes the paper.

2.2 Supervisory Control Theory

Supervisory Control Theory (SCT)(RAMADGE; WONHAM, 1989) has been developed in recent decades as an expressive framework for the synthesis of control for Discrete-Event systems (DES). According to SCT, the behaviour of a DES may be represented by sequences of events corresponding to ordered execution of activities. Among all possible sequences of events and due to the process rules and constraints, some sequences of events are desirable while other sequences are not since they violate these rules or constraints. Instead of defining a priori a specific sequence of events to be enforced in order to satisfy the constraints, the core concept of SCT is to design a supervisor that, following the sequence of events while the process evolves, specifies which events cannot occur in order to not violate the constraints. Thus, after the occurrence of an event the system (or the DES) decides which event will occur among those that are not disabled by a supervisor. SCT provides algorithms that, based on a process model considering all feasible event sequences and the associated constraints, allow one to design a supervisor whose control action imposes a minimally restrictive behaviour over a DES under consideration.

SCT is based on automata and formal language theories. Usually, a composed system is represented by a set of automata as $\{G_i|i \in I\}$, where $i \in I$ identifies each subsystem. Automaton G_i represents the independent behaviour of a corresponding subsystem in a high degree of abstraction. The uncoordinated or unconstrained behaviour of the entire DES is obtained by the synchronous product (CASSANDRAS; LAFORTUNE, 2008) of all subsystems as $G = ||_{\forall i \in I} Gi$. An automaton (also known as a language generator) is a structure as $G_i = (\Sigma^{Gi}, Q^{Gi}, \delta^{Gi}, q_0^{Gi}, Q_m^{Gi})$ where Σ^{Gi} is the alphabet (set) of events, Q^{Gi} is the set of states, $\delta^{Gi} : (Q^{Gi} x \Sigma^{Gi}) \rightarrow Q^{Gi}$ is the state transition function (in general, partially defined), q_0^{Gi} is the initial state, and $Q_m^{Gi} \subseteq Q^{Gi}$ is the set of marker states. An automaton state represents that a certain activity is being performed or that the subsystem is idle. Events represent the beginning and (un)successful execution of such activity. One may differentiate some states to give them a special meaning by grouping them in a set of marked states. In SCT marked states are those representing accomplishment of activities.

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

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

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

A supervisor is a map from the closed language of G to a subset of events to be enabled $S : L(G) \rightarrow 2^{\Sigma}$. A supervisor may be represented by an automaton and an output map $\{\Upsilon\} = (S, \Phi)$, where $S = (\Sigma^S, Q^S, \delta^S, q_0^S, Q_m^S)$. Automaton S is driven by occurrence of events in DES, and output map $\Phi : Q^S \rightarrow 2^{\Sigma_c}$ specifies the subset of controllable events that must be disabled as a correspondence of the active state of automaton S. The action of a supervisor includes disabling controllable events and unmarking sequences of events. Algorithms provided by SCT allow the formal synthesis of automaton Υ/G, which represents the optimal behaviour of G under supervision of Υ, where $L(\Upsilon/G) \subseteq L(G)$ and $L_m(\Upsilon/G) \subseteq (L(\Upsilon/G) \cap L_m(G))$. This behaviour is named supremal controllable sublanguage of $L_m(C)$ with regard to G and it is usually represented as $supC(E, G)$. Whenever $L_m(\Upsilon/G)$ is a proper subset of $(L(\Upsilon/G) \cap L_m(G))$, Υ is a marker supervisor, i.e., there are sequences of events corresponding to accomplished tasks in the uncoordinated behaviour of G that are no longer considered accomplished tasks under the action of the supervisor. Typically, the automaton representing a supervisor is automaton Υ/G itself. (MINHAS, 2002) and (SU; WONHAM, 2004) provide algorithms to obtain a reduced representation of supervisor Υ as a new pair (S_r, Φ_r) where automaton S_r has a smaller number of states than Υ/G and it provides the same control action.

In a monolithic approach, a single global supervisor is synthesised to cope with all constraints. Necessary and sufficient conditions for the existence of supervisors are established in (RAMADGE; WONHAM, 1987). According to Local Modular Control (LMC) (QUEIROZ; CURY, 2000), an extension of the SCT, instead of synthesizing a single global supervisor that satisfies the entire set of constraints, a local supervisor must be synthesized for each constraint in $\{C_j|j \in J\}$. This leads to a set of local supervisors $\{\Upsilon_j|j \in J\}$. Synthesis of local supervisor Υ_j is performed considering corresponding local constraint Cl_j and its corresponding local plant Gl_j. A local plant is obtained performing the synchronous product of only those subsystems in $\{G_i|i \in I\}$ sharing some event with corresponding constraint and local constraint is obtained performing the synchronous product of automata representing corresponding constraint and local plant ($Cl_j = Gl_j||C_j$). Automaton Υ/Gl_j represents the optimal behaviour of local plant Gl_j under supervision of corresponding local supervisor Υ_j. If at least one local supervisor in set $\{\Upsilon_j|j \in J\}$ disables an event, the event is disabled in G. A sequence of events is recognized as an accomplished task if all local supervisors agree with.

A limitation of LMC is that the behaviour obtained under the action of all local supervisors may fail to be non-blocking, even if each modular supervisor is non-blocking. Blocking in SCT occurs when all possible ways of continuing a sequence of events never lead to a marked state. After synthesis of all local supervisors, it is necessary to verify whether their control actions are free of conflicts. One way is confirming $\Upsilon/G = ||\forall j \in J\Upsilon/Gl_j$. In the worst case, such verification involves the same complexity as that found during synthesis of the global supervisor (QUEIROZ; CURY, 2000). If this property is verified, the behaviour obtained under the action of the entire set of local supervisors is identical to the behaviour obtained under the action of a global supervisor.(WONG; WONHAM, 1996) proposes how to proceed if such property is not verified.

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

- SCT uses automata as formalism to represent activities, constraints and the resulting supervisor. This is a formal and explicit way of representing them;
- Using SCT, from modelling to synthesis and visualization, the formal notation is always the same (automata), without the need to convert from one notation to another (in DECLARE is necessary to convert LTL formulas to automata);

- In SCT the state of each activity as well as each constraint may be easily visualized and understood by the end user at run-time;
- SCT provides algorithms to perform a formal synthesis of supervisor (or the admissible language of a constraint based process) instead of the usual manual and heuristic procedures;
- The obtained solution is minimally restrained and also dead-lock free;
- New control actions may be rapidly and formally designed when modifications, such as redefinition of constraints or activities arrangements, are necessary;
- The constraint-based processes can be made to behave optimally with respect to a variety of criteria, where optimal means in minimally restrictive way (concern to the admissible language of the process). This is a very strong characteristic of the proposed approach. As far as we know, there is no approach that offer a better solution related to the admissible language.

2.3 Modeling activities of constraint-based processes

The theory presented at previous section, together with a method of control implementation (VIEIRA; CURY; QUEIROZ, 2006), have successfully been employed on the actual control of DES with characteristics of the manufacturing industry (SILVA et al., 2011) (DIOGO et al., 2012). In order to provide full support to control implementation in the context of constrain-based processes, we propose first to analyze the modeling of activities and constraints. This section presents how we propose to represent activities of constraint based processes so that they may be coordinated by supervisors obtained applying SCT.

Suppose a process with a set of associated activities $A = \{a_i | i \in I\}$ where I is a set of index uniquely identifying each activity. We propose that each activity is modelled as a corresponding automaton $A_i = (\Sigma^{Ai}, Q^{Ai}, \delta^{Ai}, q_0^{Ai}, Q_m^{Ai})$, as shown in Fig. 2(a), resulting in set $\{A_i | i \in I\}$. States on this automaton mean that activity is being performed (state 1) or is not being performed (state 0). Transition from state 0 to state 1 is due to event *start activity ai* (*si*); transition from state 1 to state 0 is due to occurrence of event *successfully complete activity ai* (*ci*) or *cancel activity ai* (*xi*). In SCT marked states are those representing accomplishment of tasks, a state represented with a double line is a marked state. In this model *si* is a controllable event while *ci* and *xi* are uncontrollable events. It means that starting an activity by a resource may be disabled by a supervisor. However, once it is under execution a supervisor is not allowed to avoid it to be *cancelled* or *completed successfully*. The set of events of automaton A_i is $\Sigma^{A_i} = \{si, ci, xi\}$. Considering the entire set of activities, the whole set of events is $\Sigma = \bigcup_{\forall i \in I} \Sigma^{Ai}$. It can be seen that all pairs of automata in $\{Ai | i \in I\}$ have disjoint alphabets of events ($\forall p, q \in I, \Sigma^{Ap} \cap \Sigma^{Aq} = \emptyset$),

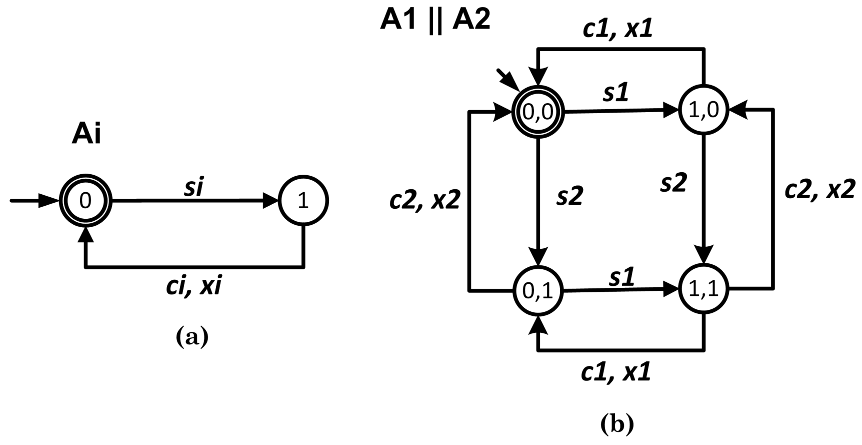


Figura 2 – Automata representing: (a)activity model and (b) uncoordinate behaviour model of two activities

so this is a product system representation. If desired, a more detailed automaton may be employed, including more states and events, it is also possible to apply a different interpretation of events' controllability. For example, it is possible to consider the activity life cycle as stated in (HOFSTEDE et al., 2010) and (REICHERT; WEBER, 2012a). The corresponding automaton may include states as *allocated*, *suspended*, *failed*, and events as *suspend*, *resume*, *allocate*, as shown in (SANTOS et al., 2012) and (SANTOS et al., 2013). The modeler has to choose which states and events will be considered based on the relevant constraints to be imposed to the process under consideration.

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

2.4 Modeling constraints

According to (AALST et al., 2011), three main categories of constraints may be identified. One category focuses on ensuring that each process instance is performed under specific ordering of activities. The second category focuses on managing the allocation or usage of

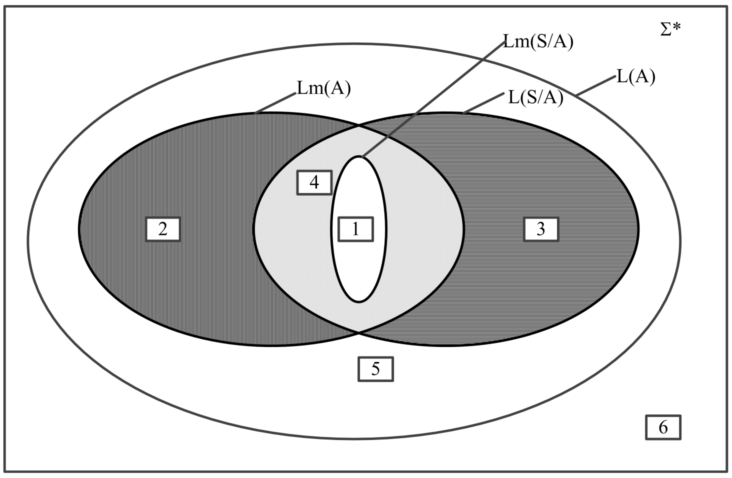


Figura 3 – Venn diagram of languages relation

resources that perform such activities. The third one focuses on the attributes of a process instance. SCT may be employed to synthesize supervisors to enforce these constraints. This paper is restricted on modelling activities compatible with constraints on the first category. In this section we describe the procedure to represent a constraint using automata.

Consider automaton **A** representing the uncoordinated behaviour of a set of activities $\{ai|i \in I\}$ of a process. Language $L(A)$ represents all sequences of events that may be performed by these activities without any constraint, and $L_m(A)$ is a subset of $L(A)$ representing accomplished activities. The basic premise is that a process contains sequences of events in $L(A)$ that are not acceptable because they violate some constraint. It is also possible that certain states must be forbidden since they represent an unauthorized concurrent execution of activities. These sequences and states must be avoided. Also, it is possible that a sequence of events in $L_m(A)$ does not correspond to an accomplished task when the process instance is performed under supervision. Thus, that sequence needs to be unmarked by supervisor. Consider automaton S/A, such that $L_m(S/A) = supC(C, A)$, the one that recognizes the supremal controllable language of constraint activities. (REICHERT; WEBER, 2012a) define a supported traces as a sequence of events complying with all mandatory constrains. This definition complies with the definition of a sequence of events belonging to $L_m(S/A)$. Fig. 3 shows the languages relation: the region 1 includes sequences of events belonging to $L_m(S/A)$ (or supported traces); the region 2 includes sequences w such that $w \in L(A)$, $w \in L_m(A)$, $w \not\ni L(S/A)$, $w \not\ni L_m(S/A)$; the region 3 includes sequences $w \in L(A)$, $w \not\ni L_m(A)$, $w \in L(S/A)$, $w \not\ni L_m(S/A)$, the region 4 includes sequences $w \in L(A)$, $w \in L_m(A)$, $w \in L(S/A)$, $w \not\ni Lm(S/A)$.

To formally obtain the supervisor that restrains the uncoordinated behaviour of activities it is necessary to express constraints in terms of automata. Usually, each constraint is represented as an automaton resulting in set $\{C_j|j \in J\}$, where J is a set of index uniquely identifying each constraint. When a process instance is performed under supervision of a

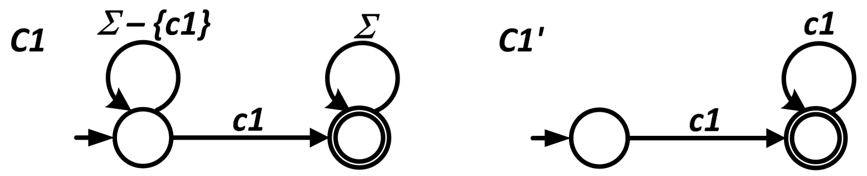


Figura 4 – Automata representing constraint *existence(a1, 1)*

set of supervisors obtained employing SCT, the related constraints will never be violated and there will always be at least one sequence of events leading to a marked state, i.e., there will always be the possibility of accomplishing a task.

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

The *existence(a1, 1)* model requires that *activity a1 must occur at least once at every trace* (REICHERT; WEBER, 2012a)(PESIC, 2008). In order to facilitate the understanding of our approach, we rewrite this constraint to *activity a1 is successfully completed at least once*. Fig. 4 presents two possibilities of modelling this constraint. The first possibility is through automaton **C1**. Initial state of **C1** has a self-loop labelled as $\Sigma - c1$, meaning that the occurrence of any event belonging to Σ but $c1$ keeps this state as the active one. The active state is state 1 only after occurrence of event $c1$. This remains the active state despite the occurrence of any event, as there is a self-loop labelled as Σ. Since the only marked stated is state 1 then accomplishing a task is recognized only after first occurrence of event $c1$. Alphabet of events of this automaton is the whole set of events $\Sigma^{C1} = \Sigma$. Modelling a constraint through an automaton whose alphabet of events is Σ has the advantage of clearly presenting the occurrence of all possible events. In the case one employs this automaton as a constraint under the LMC approach, then the corresponding local plant is automaton A and does not take advantage of this approach in reducing computational complexity to synthesize a corresponding local supervisor. Considering definition of synchronous product and algorithms for the synthesis of supremal controllable language $supC((A\|C1), A)$ (WONHAM, 2011), a more efficient representation of a constraint is through an automaton employing only strictly necessary events.

The second possibility of representing *existence(a1, 1)* model is through automaton **C1'**, where the alphabet of events is $\Sigma^{C1'} = c1$. In this case it is implicit that any event that does

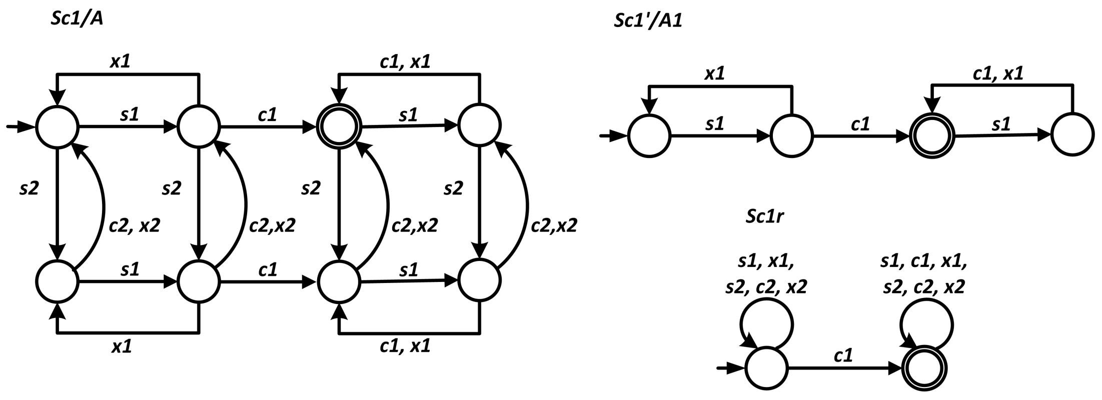


Figura 5 – Automata Sc1/A, Sc1'/A1 and Sc1r

not belong to $\Sigma^{C1'}$ is always allowed to occur in accordance with automaton representing the uncoordinated behaviour as automaton **A**. Adopting **C1'** as a constraint, results that corresponding local plant is automaton **A1** alone. Thus computational complexity on the synthesis of local supervisor and number of states of automaton representing it will be smaller than in the first possibility. Fig. 5 presents automata $Sc1/A$ and $Sc1'/A1$, where $L_m(Sc1/A) = supC((A\|C1), A)$ and $L_m((Sc1'/A1) = supC((A1\|C1'), A1)$ may be employed as supervisors enforcing constraint *existence(a1, 1)* over the set of activities $A = \{a1, a2\}$. Control action of supervisors obtained in both possibilities will be equivalent. Such control action is only unmarking sequences of events (recognizing accomplished tasks), it will not disable events. Applying supervisor's reduction algorithms (WONHAM, 2011) on these supervisors results automaton $Sc1r$, also shown in Fig. 5. It can be seen from automaton **A** (Fig. 2) that sequences $w1 = \varepsilon$ (the empty sequence of events), $w2 = s1x1$, $w3 = s2c2$, $w4 = s1c1$, among others, lead from initial state back to it and this is a marked state so $w1$, $w2$, $w3$, $w4 \in L_m(A)$. It can also be seen from automaton $Sc1/A$ and $Sc1'/A1$ that while sequence $w4$ (region 1 according to Fig. 3) leads from initial state to a marked state the same is not true for sequences $w1$, $w2$ and $w3$; this means that $w4 \in L_m(Sc1/A)$ but $w1$, $w2$, $w3 \not\ni L_m(Sc1/A)$ (region 4 according to Fig. 3). While $w4$ is a supported trace, as defined by (REICHERT; WEBER, 2012a), $w1$, $w2$ and $w3$ are unsupported ones. Language $L_m(Sc1/A)$ contains all possible sequences of events recognized as accomplished tasks under supervision. Thus, it contains all supported traces.

According to (REICHERT; WEBER, 2012a) and (PESIC, 2008), the constraint *response(a1, a2) states that if a1 is executed, a2 needs to be executed afterwards (but not directly after).* We may rewrite it replacing *executed* by *completed.* Fig. 6 shows the automaton **C2** as a possible model for representing this constraint employing only strictly necessary events, where $\Sigma^{C2} = \{c1, c2\}$. In this automaton the state transition function is defined with the occurrence of all events in Σ^{C2} at every state meaning that they are always allowed to occur. While state 0 is a marked one, state 1 is not, meaning that sequences of events leading to state 1 are not considered as accomplished tasks because there has been at least one

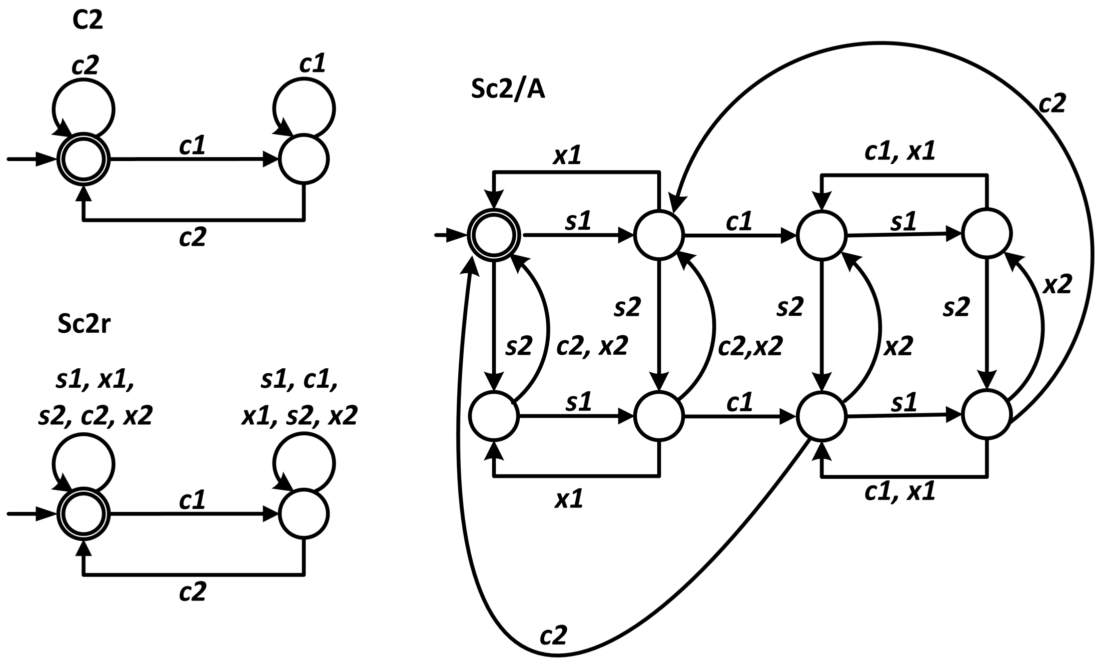


Figura 6 – Automata C2, Sc2/A, and Sc2r

occurrence of $c1$ that was not followed by $c2$. In this case automaton **A** represents the local plant since that constraint employs events of every activities. Automaton $Sc2/A$, where $L_m(Sc2/A) = supC((A\|C2, A)$, is a possible supervisor's representation of a supervisor enforcing this constraint, and automaton **Sc2r** is a reduced representation of it. Again, supervisor's control action is only unmarking sequences of events: the output map is always empty, i.e. $\forall q \in Q^{Sc2/A} \rightarrow (\Phi(q) = \emptyset)$. As shown in (REICHERT; WEBER, 2012a), $< a1, a2 >$, $< a1, a1, a1, a2 >$ and $< a2 >$ are supported traces while $< a1 >$ is an unsupported trace. In these cases it is only considered an abstraction of activities, i.e., only considered a single event representing the execution and completion of an activity. Also, it is not considered the overlapping of activities, i.e., activities are only sequentially executed. Sequences of events that may represent traces with *start* and *complete* events, may be $w5 = s1c1s2c2$, $w6 = s1c1s1c1s1c1s2c2$, $w7 = s2c2$, $w8 = s1c1$. Considering trace $< a1, a2 >$ there are many other sequences of events, including overlapping activities. For instance $w9 = s1x1s1x1s1c1s2c2$, $w10 = s1s2c1c2$, $w11 = s1c1s2x2s2x2s2c2$, all traces belonging to $L_m(Sc2/A)$. While $w8$ is an unsupported trace (region 4 in Fig. 3), $w5$, $w6$, $w7$, $w9$, $w10$ and w11 are supported traces (region 1 in Fig. 3).

(REICHERT; WEBER, 2012a) and (PESIC, 2008) also present the constraint *precedence(a1, a2)* as *activity a2 needs to be preceded by activity a1.* We may rewrite it as *a2 can be completed only after a1 has been completed at least once.* Fig. 7 shows the automaton **C3** as a possible model for representing this constraint. Notice that both states are marked, meaning that corresponding supervisor will not unmark sequence of events. Since $\Sigma^{C3} = \{c1, c2\}$ and state transition function is not defined with the occurrence of $c2$ at state 0 than this event cannot occur at this state, i.e. prior to first occurrence of

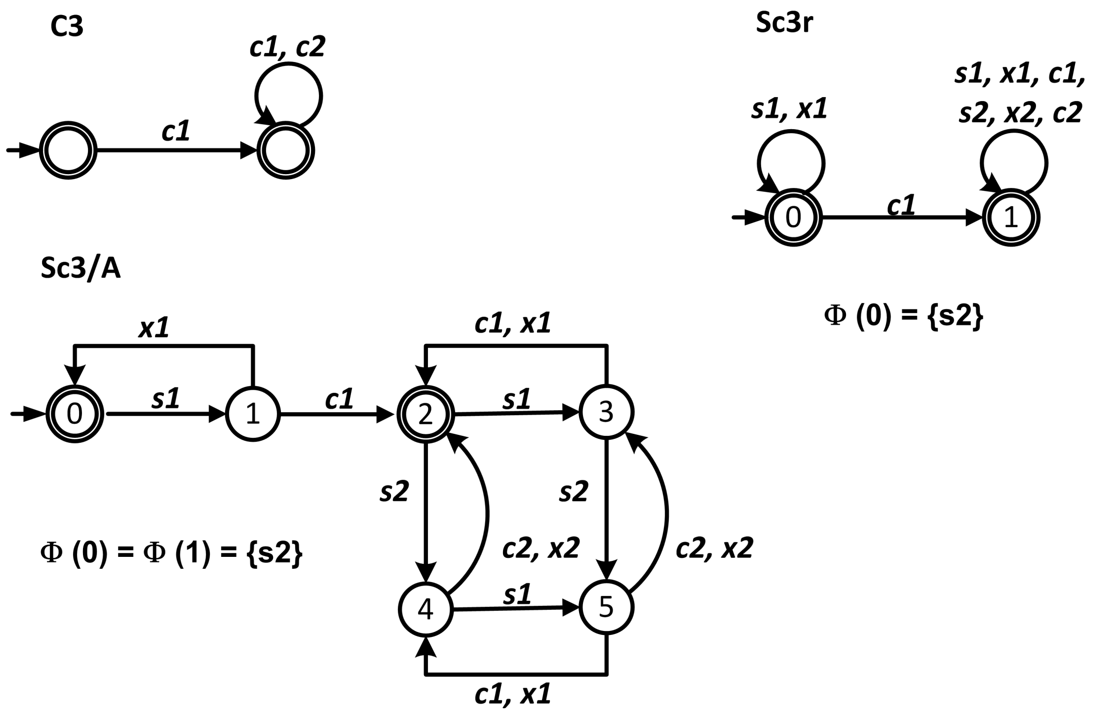


Figura 7 – Automata C3, Sc3/A, and Sc3r with c2 as an uncontrollable event

$c1$. Automaton $Sc3/A$, where $L_m(Sc3/A) = supC((A \parallel C3), A)$, is a possible supervisor's representation. In this case supervisor's control action is only disabling controllable events, and corresponding output map specifies that event $s2$ is disabled at states 0 and 1 $(\Phi(0) = \Phi(1) = \{s2\}, (\forall q \in Q^{Sc3/A}, q \neq 0, q \neq 1) \rightarrow (\Phi(q) = \emptyset))$. The aim of this supervisor is to avoid occurrence of event $c2$ prior to the first occurrence of $c1$. Since $c2$ is considered to be an uncontrollable event then the supervisor needs to take an anticipatory action disabling $s2$ (a controllable event).

Fig. 8 presents automata $Sc3/A'$ and $Sc3r'$ considering $c2$ as a controllable event. Notice that automaton $Sc3/A'$ has two extra states due to occurrence of event $s2$ from state 0 and from state 1. Also, supervisor's control action is disabling occurrence of event $c2$ at these extra states (6 and 7) instead of event $s2$ at states 0 and 1. As shown in (REICHERT; WEBER, 2012a), $< a1, a2 >$, $< a1, a2, a2, a2 >$ and $< a1 >$ are supported traces while $< a2 >$ is an unsupported trace. Sequences of events that may represent such traces are, respectively, $w12 = s1c1s2c2$, $w13 = s1c1s2c2s2c2s2c2$, $w14 = s1c1$, $w15 = s2c2$. While $w15$ is an unsupported trace (region 2 in Fig. 3), $w12$, $w13$, $w14$ are supported traces (region 1 in Fig. 3). Considering constraint precedence $(a1, a2)$ and $c2$ as an uncontrollable event, sequences in region 3 in Fig. 3 are $w16 = s1$ and $w17 = s1c1s2$; a sequence in region 5 is $w18 = s2$ and a sequence in region 6 is $w19 = c1s1$.

According to (REICHERT; WEBER, 2012a), constraint-based process models focus on what should be done by describing the activities that may be performed and the constraints prohibiting undesired execution behaviour. In the present paper we restrict our focus on

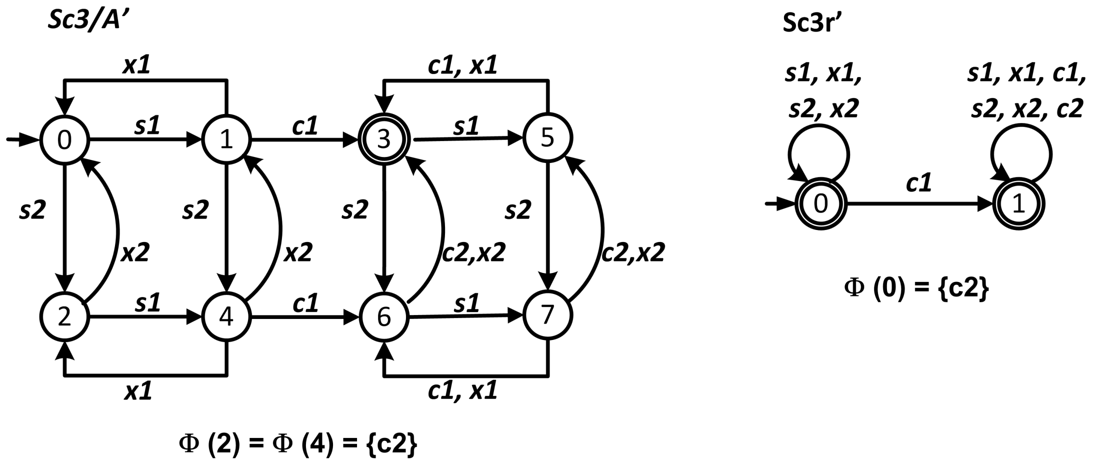


Figura 8 – Automata Sc3/A', and Sc3r' with c2 as a controllable event

constraints aiming to ensure that each process instance is performed under an ordering of activities and we use the same principle as proposed in (PESIC, 2008) and (SCHAIDT et al., 2013), where is considered four groups of constraints: (1) existence, (2) relation, (3) negation and (4) choice. Existence models specify how many times or when one activity may be executed. Relation models define some relation between two (or more) activities. Negation models define a negative relation between activities. Choice models can be used to specify that one must choose between activities. Because the space limitation of this paper, we only present some models of each group. Fig. 9 to Fig. 12 shows some constraint models using automata.

2.5 Application example

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

The PMBOK provides three stages for each process: Inputs, Tools and Techniques, and Outputs. The inputs to this process are the documents *Project Charter* (PC) and *Stakeholder Register* (SR). The tools and techniques adopted for implementing this model are: interviews, focus groups, facilitated workshops, questionnaires and surveys, prototypes and brainstorm. The outputs suggested by PMBOK are *Requirements Documentation* (RD), *Requirements Management Plan* (RMP) and *Requirements Traceability Matrix* (RTM). For this work we selected the output *Documentation Requirements.* Thus, activities under

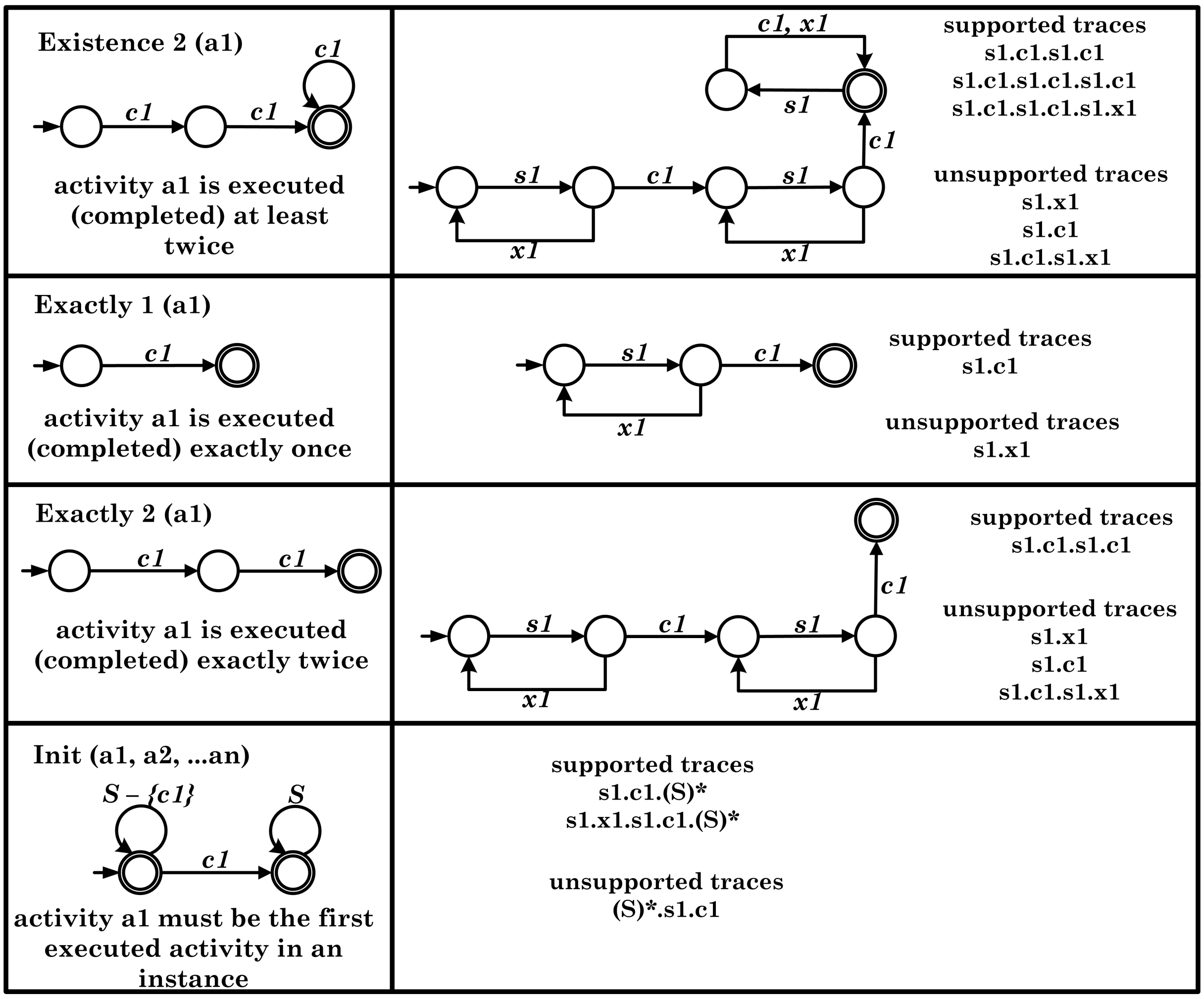
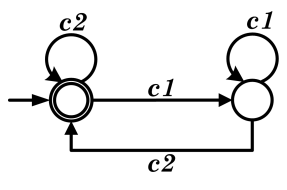


Figura 9 – Automata representing the group existence

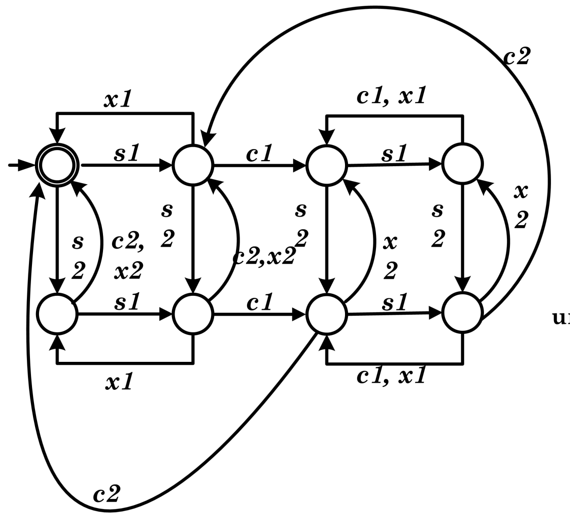
control are *Review Project Charter and Stakeholders Register* ($a1$), *Brainstorm* ($a2$), *Focus Groups* ($a3$), *Facilitated Workshops* ($a4$), *Questionnaires and Surveys* ($a5$), *Interviews* ($a6$), *Prototypes* ($a7$), and *Requirements Documentation* ($a8$). We assume each activity *ai* (i=1,. . . ,8) is modeled as an automaton shown in Fig. 2 (a).

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

Response (a1, a2)



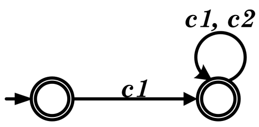
every time activity a1 executes, activity a2 has to be executed after it. In this case a2 does not have to execute directly after a1, and another a1 can be executed after the first a1 and the subsequent a2



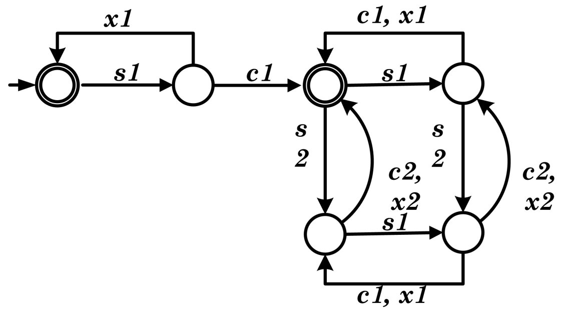
supported traces
s1.c1.s2.c2
s1.c1.s1.c1.s2.x2
s2.c2

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

Precedence (a1, a2)



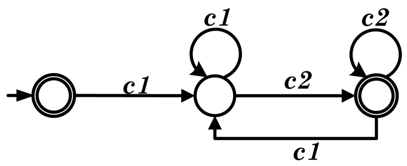
activity a2 needs to be preceded by activity a1, i.e., it specifies that activity a2 can be executed only after activity a1 has been executed



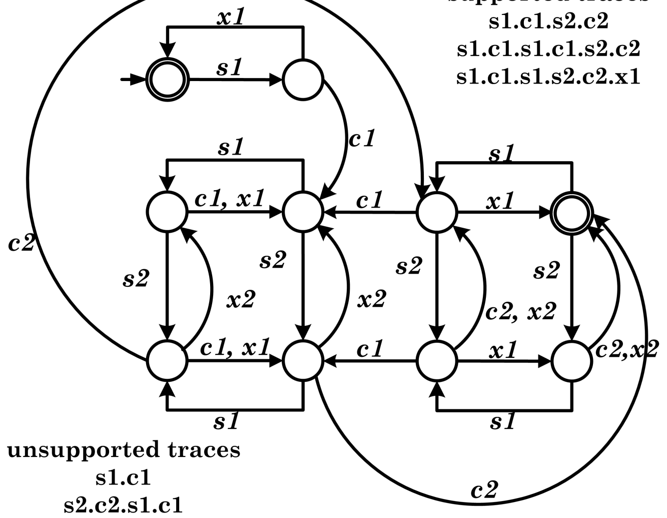
supported traces
s1.c1
s1.c1.s2.c2.s1.c1
s1.x1
s1.c1.s2.s1.c1.c2

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

Succession (a1, a2)



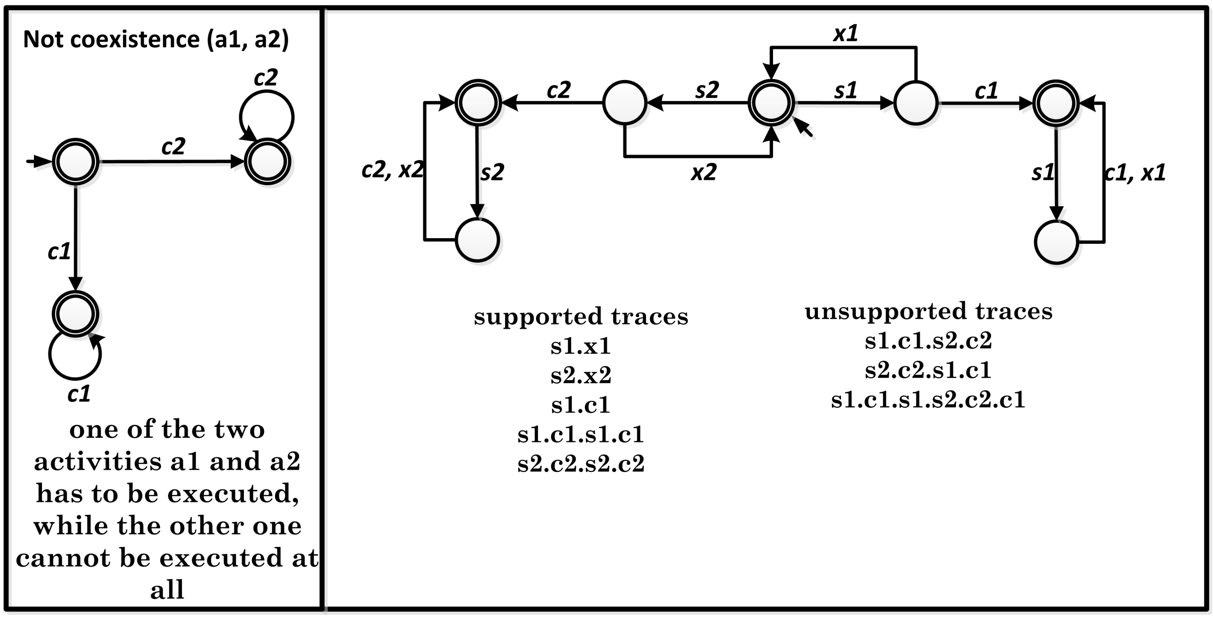
both response and precedence constraints have to hold between the activities a1 and a2



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

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

Figura 10 – Automata representing the group relation

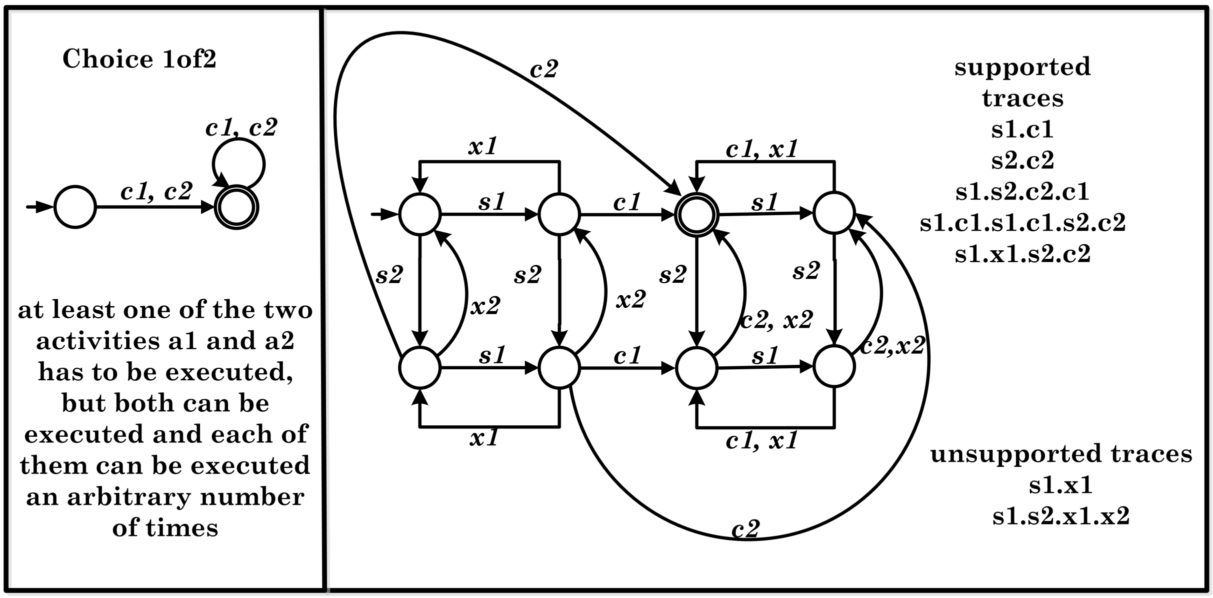


one of the two activities a1 and a2 has to be executed, while the other one cannot be executed at all

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

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

Figura 11 – Automata representing the group negation



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

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

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

Figura 12 – Automata representing the group choice

2.6 Executing the supervisory control

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

When the preconditions for executing a particular activity are met during run-time, a new instance of this activity is created. Hence, an activity instance represents a single invocation of an activity during the execution of a particular process instance. Particularly, when a human activity becomes enabled during the execution of a process instance, the Process-Aware Information System (PAIS) first determines all resources qualifying for this activity instance. For each potential resource, a work item referring to the activity instance is created and added to his worklist. Work items related to a particular activity instance may be added to different user worklists. Generally, a worklist comprises all work items currently offered to, or processed by, a user (REICHERT; WEBER, 2012a).

Generally, resources (process participants or users) interact with a PAIS via worklists. When a resource allocates a work item from his worklist, all work items related to the same activity instance are removed from the worklists of other resources. Further, the resource to whom the work item is allocated may then trigger the start of the application service associated with the corresponding activity instance. The supervisory control must ensure that activities are executed considering the specified constraints during run-time (REICHERT; WEBER, 2012a). Fig. 13 shows three process instances PI_1, PI_2 and PI_3 running on the process presented in Section 2.5. The depicted worklists of resources John, Paul and George comprises work items related to these three process instances.

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

- Process instance PI_2: Activities $a1$, $a2$ and $a4$ have already been completed but they are still enabled. This is because they may be executed any number of times. Notice that these activities have been added to the worklists of John and Paul. Activities $a3$ and $a7$ are disabled by the supervisory control and the activities $a5$,$a6$ and $a8$ are enabled and they have not been executed yet. These activities have been added to the worklists of Paul and George.

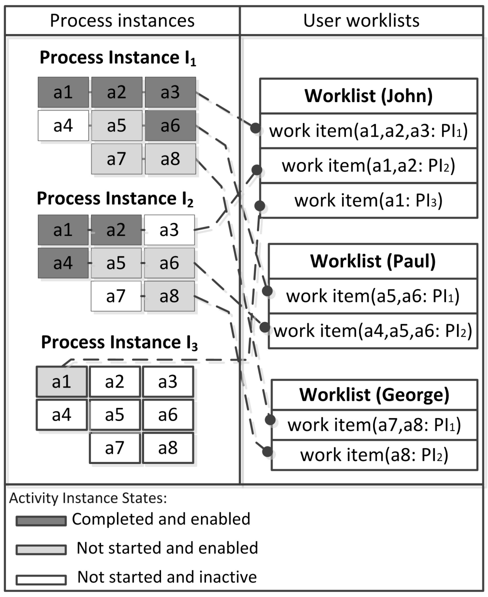


Figura 13 – Project management process, process instances and users worlists

- Process instance I_3: Only activity $a1$ is enabled (and it has not been executed yet). Activities $a2$ to $a8$ are disabled by the supervisory control. Notice that in this state the process has not been initiated. Activity $a1$ has been added to the worklist of John.

According to our approach, during the execution of activities the supervisory control has to disable activities in order to not violate the constraints. In terms of run-time environment, the supervisory control cannot add an activity in any worklist if such activity is disabled by the supervisor. As long as an event (si, ci or xi) occurs, the state of the supervisor is updated and a new control action (a list of disabling activities) is established. Notice that in our approach the activities may be executed without overlapping (sequentially executed only) and with overlapping (executed in parallel). Fig. 14 illustrates the execution of the process shown in Section 2.5 considering overlapping activities for a specific process instance.

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

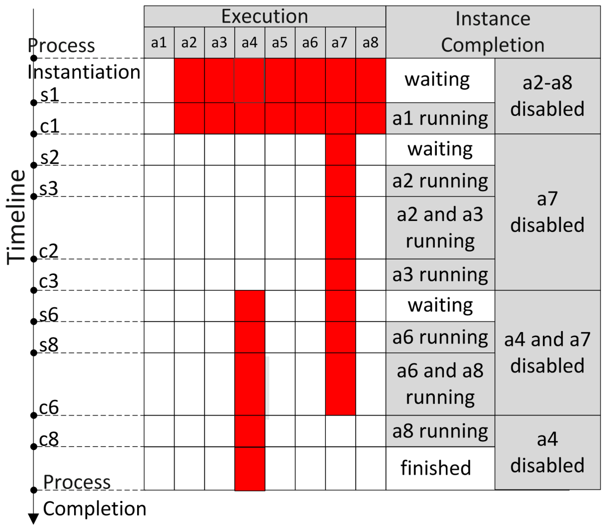


Figura 14 – Executing the project management process under supervision

$a7$ becomes enabled. However, the process instance is finished when the activity $a8$ is completed.

2.7 Conclusion

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

3 Selection of process variants from pre-specified processes based on supervisory control theory

Abstract

Process models are often reused in different contexts, resulting in a large number of related process model variants. Such process variants pursue the same or similar business objective, but may differ in their logic (i.e., process logic) due to varying application context at either design time or run-time. We propose a formal procedure to support the selection and configuration of process variants. Our approach is based on Supervisory Control Theory (SCT), which is a formal way to build supervisors for discrete-event systems. In our approach, a questionnaire is used to support configuration and selection of a process variant, and a formal procedure links the questionnaire to a set of constraints. This questionnaire has a set of questions and a set of possible answers associated to each question. After questionnaire answering, we propose a formal procedure to select a process variant. Our idea is that a selected answer determines a set of tasks that have to appear in the process model. The whole procedure is founded on SCT approach.

3.1 Introduction

There is a need for enterprises to adapt their processes in different application environments in a fast and flexible way. However, designing business process models from scratch is a time-consuming and costly task, besides process models usually vary over time which makes this task even more challenge (REICHERT; WEBER, 2012a). Thus, the reuse of process models is a crucial task to maintain competitiveness in business environments. A process model may be used in different application contexts. They can also set some variable attributes which are related to different circumstances. To reuse a process model in different contexts can result in a wide range of related process model variants which belongs to the same process family. These process variants are connected to the same business objectives and they have several common points but also there are differences due specific conditions of each context, for example, some activities can be required for a context but entirely unnecessary for other.

Variability is the type of flexibility that permits a process model to be configured according to a specific circumstance (ROSA et al., 2009). Process variability can be required in different domains when processes need to be handled in function of a business process context resulting different process variants (HALLERBACH; BAUER; REICHERT, 2010b). Process variants are usually derived from the same process model and the concrete sequence of actions vary for each variant (HALLERBACH; BAUER; REICHERT, 2008b). At least four aspects that can generate a process variant: product and services, regulations and laws, type of clients, and time (REICHERT; WEBER, 2012a). Product and service variability is required because there can be concrete product variant in a only business. Differences in regulations in different countries and regions can derive different process variants at the same business. Variability might be also required from different types of customers (premium or standard, for example) and due to temporal differences (seasonal changes, for example). The real variant can often only be defined while the process is executed, but the general model from which each variant shall be derived is known previously. A healthcare process for emergency patient treatment can be an example of process variability. Before each patient is treated an evaluation of his general condition is done resulting in a setting of which actions must be executed and which must not be executed from a general process model, the resultant is a process variant (LU et al., 2009) .

It is too expensive for companies to design and implement standardized business processes for each context of the real world so there is a large interest in gathering common process knowledge to use it as a reference process model and from this one can derive all variants conformance with each context of application (HALLERBACH; BAUER; REICHERT, 2010a; HALLERBACH; BAUER; REICHERT, 2008a; HALLERBACH; BAUER; REICHERT, 2008c). So it is necessary a modeling approach to capture and set the variability in a process model, this approach must be able to represent a family of process variants in a compact, reusable, and maintainable way and it should allow for easily configure a process family to a process variant that adequately represents the requirements of a specific application environment (HOMAYOUNFAR, 2012).

In this context, the reference process model concept was developed. The reference process model is reused and adapted in several ways to achieve different goals, by producing many process variants from it. Thus, the reference process model take the form of libraries of process models structured as hierarchies providing an alternative to designing process models from scratch. Some examples of reference process models are the Supply Chain Operations Reference (SCOR) model, ITIL for IT service management, SAP reference models, or medical guidelines for patient treatment (FAQUIH; SBAÏ; FREDJ, 2015; SBAI; FREDJ; KJIRI, 2012; FAQUIH; SBAÏ; FREDJ, 2014).

However, in general reference process models do not capture possible variations in a systematic manner, so the variation points and configuration decisions are not represented in these models. As result, analysts are given little guidance when model elements need to be removed, added or modified to meet a given set of requirements. Besides, process modelling tools do not adequately support the handling of such process variants. These must be prespecified either in terms of separate process models or by using one process model with conditional branching. However, both approaches can result in model redundancies that significantly aggravate model maintenance, and thus turn it into a time-consuming and error-prone task (FAQUIH; SBAÏ; FREDJ, 2014; REICHERT; WEBER, 2012a; KAYMAK et al., 2012).

In healthcare area the variability is very common due to certain characteristics of their processes. In general the healthcare processes have the following characteristics:

- highly dynamic due to the constant arising of new drugs, procedures, treatments and diseases healthcare processes are executed according to a wide range of distributed activities, performed by the collaborative effort of professionals with different skills, knowledge and organizational culture;
- highly variable due to its non-repetitive character, and to its non-deterministic order of execution As a consequence of these is that it is not precisely known what happens in a healthcare process for a group of patients with the same diagnosis. Considering a group of patients with the same condition, a number of different examinations and treatments may be required and the order in which they are conducted may greatly vary (MANS et al., 2008; GUPTA, 2007; HOMAYOUNFAR, 2012; LENZ; REICHERT, 2007).

This paper proposes an approach that builds a bridge connecting specialists of different fields of knowledge, a process specialist (PS) and an information technology (IT) specialist, such that, acting together and employing a common language they will be able to represent a reference process model and select variants of it. Applying this approach will result in a set of documents that will formally register requirements specifications of a project involving a process' specialist and an IT' specialist employing a common language to both. In a further step an IT's specialist will be able to translate this specifications into a set of automata allowing him to formally synthesize a reference process model and its variants employing the SCT. This results allows one to develop an information system aiming to manage and to supervise the execution of process variants instances. This last step is out of the scope of this paper.

The aim of this work is to propose an approach to select a process variant from a reference process model by using a questionnaire and a formal procedure based on Supervisory Control Theory (SCT) (RAMADGE; WONHAM, 1989; RAMADGE; WONHAM, 1987). A questionnaire is used to support configuration and selection of a process variant, and a formal procedure links the questionnaire to a set of constraints. A process variant is selected using the algorithms proposed by SCT approach. This questionnaire has a set of questions and a set of possible answers associated to each question. Basically, we add control over questions sequence in order to obey a certain set of constraints. Thus, the proposed approach restrains the space of allowed answers in order to prevent users from making inconsistent selections during configuration. After questionnaire answering, we propose a formal procedure to select a process variant. Our idea is that a selected answer determines a set of tasks that have to appear in the process model.

The present paper is organized as follows: Section 2 presents the literature review related to the proposed approach. Section 3 describes the Supervisory Control Theory, as the fundamental concept of our approach. Section 4 presents the running example used to explain the proposed approach. Section 5 describes in detail the formal procedure to select a process variant from a reference process model. Section 6 concludes the paper.

3.2 Related Works

Several studies have been interested in different aspects of business processes variability. We surveyed process variability research to discover the main approaches in this area. To perform the research, were select papers containing the words ‘configurable process model’, ‘business process variability’, ‘business process flexibility’, ‘process family’, ‘reference process model’ and ‘process variant’ as key-words or in the abstract. These works are presented in the next two Tables. Table 1 present researches that approached topics related to the (re)design, management, modeling, configuration and validation of the Configurable Process Models (CPM).

Koschmider e Oberweis (2007) developed an algorithm for determining linguistic similarities between business process model variants, thus facilitating process redesign. Kumar and Yao, 2012 in Kumar e Yao (2012) proposed the design and management of flexible process variants by applying business rules to a generic process template, which describes a very basic and general process schema.

The CPM management and configuration were the topics most discussed between the papers analyzed. Gottschalk, Aalst e Jansen-Vullers (2007) developed an approach for configuring and managing process variant named Propov, which provides an operational approach for managing process variants based on a single process model. In this approach, process variants can be configured by applying a set of high-level change operations (e.g., to insert, delete or move process fragments) to a given process model. Reijers, Mans e Toorn (2009) proposed simplify business process model management by combining the business process model into aggregate models where the common part is included only once and the unique parts from each of the separate models are preserved.

Focusing on the configuration, Rosa et al. (2009) developed a questionnaire-based framework for configuring reference process models. In this way, each question refers to a set

Tabela 1 – Researches on business process variability aspects

Authors	Description
Becker et al. [18], La Rosa et al. [19], Ognjanović et al. [20], Santos et al. [21], La Rosa et al. [22], Derguech and Bhiri [23]	CPM configuration
Gottschalk et al. [24]	CPM modelling
Koschmider et al. [25], Pascalau et al. [26]	CPM (re)design
Lapouchnian et al. [27]	CPM (re)design, CPM configuration
Hallerbach et al. [28]	CPM management, CPM configuration
Thomas [29], Lu et al. [30], Reijers et al. [31], Derguech et al. [32]	CPM management
Reichert et al. [33]	CPM management, CPM modelling
Reinhartz-Berger et al. [34], Gröner et al. [35]	CPM validation
Mahmod and Chiew [36], Kumar and Yao [37]	CPM (re)design, CPM management
Pascalau et al. [38]	CPM management

of facts that can be set to true or false. Facts encode the variability of the system, e.g. optional features, values of configuration parameters, etc. The individualization of the generic system is captured by means of actions. As the questionnaire is answered, values are assigned to facts, and the resulting valuation of facts determines which actions should be performed on the generic system to derive an individualized system.

Derguech e Bhiri (2011) proposed an algorithm that allows for merging a collection of business process models to create a configurable process model. The algorithm ensures that the resulting configurable model includes the behaviors of the original business process variant by considering work nodes with identical labels and preserving the status of start and end nodes.

Regarding to the modeling of configurable process models, Reichert et al. (2009) presents an extension of the ARIS Business Architect to better cope with the high variability of business process models in practice. This extension is based in the Provop framework proposed by the authors to support the modeling and management of process variants. Gottschalk, Aalst e Jansen-Vullers (2007) presented an analysis of configuration from a theoretical perspective. Within the analysis a link is made to inheritance of dynamic behavior and previously defined inheritance concepts. By applying these concepts to process models the essence of configuration is determined, which enables the development of more mature configurable process modeling languages.

Some authors focused in the aspects related to the validation of the configurable process model. Reinhartz-Berger, Soffer e Sturm (2009) applies the ADOM (Application-based domain modelling) as a platform for organisational reference models, and introduced a validation procedure to check the compliance of the specific processes with the organisational reference model. Gröner et al. (2011) proposed a classification of interrelationships between elements of business process models and demonstrate how this classification can be used for the validation. The classification, specified in Description Logics (DL), is based on an analysis of basic workflow patterns, a set of conceptual basis for process languages.

In Table 2 are presented some studies which focuses on topics such as process families, reference process models, the syntactic and/or semantic correctness of CPM, the aspects related to the business process variability and the literature review about business process variability.

Tabela 2 – Researches on business process variability aspects

Author(s)	Process families
von der Maßen and Lichter [43], Razavian and Khosravi [50], Rolland and Nurcan [53], Nguyen et al. [56], Ayora et al. [16], Yao and Sun [59], Mechrez and Reinhartz-Berger [62]	Business process variability
Recker et al. [44], van der Aalst et al. [18], van der Aalst et al. [54], van der Aalst et al. [58]	Syntactic/semantic correctness of CPM
Reinhartz-Berger et al. [45], Rabe et al. [7], Lazovik and Ludwig [47], Reinhartz-Berger et al. [52], Li et al. [55]	Reference process models
Giese et al. [46], Schonenberg et al. [48], La Rosa et al. [49], Vergidis et al. [51], Ayora et al. [14], Torres et al. [57], Ayora et al. [60]	Process families
Gröner et al. [61]	Process families, Business process variability
Valença et al. [4], Ayora et al. [11]	Literature review

Some researches approached issues related to the process families. Gröner et al. (2013) developed a validation algorithm ensuring that each member of a business process family adheres to the core intended behaviour that is specified in the reference process model. The proposed validation approach is based on modelling and reasoning in Description Logics, variability is represented by using the Feature Models and behaviour of process models is considered in terms of control flow patterns. Ayora et al. (2013b) proposed nine patterns for dealing with changes in process families. The authors introduced a set of generic and language-independent patterns that cover the specific needs of process families. When used in combination with existing adaptation patterns, change patterns for process families will enable the modelling and evolution of process families at a high-level of abstraction.

Focusing on reference process model, some topics addressed are the discovery of the reference process model, the adaptability and the customizability of the reference among others. Reinhartz-Berger, Soffer e Sturm (2005) propose to utilize the ADOM, for specifying and applying reference models. The benefits of utilizing the ADOM approach for specifying business models are the provisioning of validation templates by the reference models and the ability to apply the approach to various modelling languages and business process views. Li, Reichert e Wombacher (2011) introduced, evaluated and compared two algorithms (heuristic and clustering) for discovering a reference process model out of a collection of block-structured process variants.

Ensuring the syntactical and/or semantic correctness of the process model is the focus of some researches. Recker et al. (2005) have shown that the application of configurable EPCs in the process of enterprise system reference model configuration leads to syntactic problems. Thus, the authors outlined a XML schema-based approach using the EPC Markup Language for the syntactical validation of reference process model configuration. Aalst et al. (2008) proposed a framework for configuring reference process models which includes a technique to derive propositional logic constraints that, if satisfied by a configuration step, guarantee the syntactic and semantic correctness of the resulting model.

Regarding the business process variability, Ayora et al. (2013a) presents an evaluation framework that allows analysing and comparing the variability support provided by existing proposals developed in the context of business process variability. Based on an in-depth analysis of several large process model repositories from various domains, the framework defines both a set of language requirements and variability support features needed for properly dealing with BP variability. Mechrez e Reinhartz-Berger (2014) proposed a two-dimensional framework that refers to granularity, namely, the variable elements, and guidance, i.e., the creation of variants at design-time. The framework is used for evaluating the expressiveness of 22 languages that support design-time variability modelling in business processes.

Finally, related to the literature review the researches addressed approaches and challenges related to process flexibility (SCHONENBERG et al., 2008; AYORA et al., 2012), techniques for capturing variability and techniques to capture the domain parameters that affect the variability (ROSA; DUMAS; HOFSTEDE, 2009), the business process modelling, analysis and optimization vergidis2008business, comparison between C-EPC and Propov approaches (TORRES et al., 2012), mapping of business process variability (VALENÇA et al., 2013) and a framework for assessing and comparing process variability approaches (AYORA et al., 2015).

Others aspects related to the business variability were also focused such as the automatic creation of CPM (JIMÉNEZ-RAMÍREZ et al., 2013), the soundness of CPM (HALLERBACH; BAUER; REICHERT, 2009; SCHUNSELAAR et al., 2012b), CPM notations (ROSA et al., 2013), to predict the complexity of CPM (VOGELAAR et al., 2011), the evolution of the CPM (SBAI; FREDJ; KJIRI, 2013), (SBAI; FREDJ; KJIRI, 2014) among others.

Nevertheless, many of these approaches have several limitations due to the low level of automation (REICHERT; WEBER, 2012a). Furthermore, an important issue concerned with business process variability is the auto-verification of business process variants obtained after configuration. Another need is to have a taxonomy for the variability of business processes to facilitate the management of research, configuration and evolution of these processes (FAQUIH; SBAÏ; FREDJ, 2015).

3.3 Supervisory Control Theory

Supervisory Control Theory (SCT) (RAMADGE; WONHAM, 1989; RAMADGE; WONHAM, 1987) has been developed in recent decades as an expressive framework for the synthesis of control for Discrete-Event systems (DES). According to SCT, the behaviour of a DES may be represented by sequences of events corresponding to ordered execution of activities. Among all possible sequences of events and due to the process rules and constraints, some sequences of events are desirable while other sequences are not since they violate these rules or constraints. Instead of defining a priori a specific sequence of events to be enforced in order to satisfy the constraints, the core concept of SCT is to design a maximal controllable language that, following the sequence of events while the process evolves, specifies which events cannot occur in order to not violate the constraints. Thus, after the occurrence of an event the system (or the DES) decides which event will occur among those that are not disabled by a supervisor. SCT provides algorithms that, based on a process model considering all feasible event sequences and the associated constraints, allow one to design a maximal controllable language that represents a minimally restrictive behavior over a DES under consideration.

Suppose a process with a set of associated activities A $= \{a_i \mid i \in \mathrm{I}\}$ where I is a set of index uniquely identifying each activity. We propose that each activity is modelled as a corresponding automaton. Each automaton is represented by a 5-tuple $\mathrm{A}_i = (\Sigma^{A_i}$, Q^{A_i}, δ^{A_i}, $q_0^{A_i}$, $Q_m^{A_i})$, where Σ_{A_i} is the alphabet (i.e., set) of events; Q^{A_i} is the set of states; δ^{A_i} : $(Q^{A_i}$ x $\Sigma^{A_i}) \rightarrow Q^{A_i}$ is the state transition function, which is typically partially defined; $q_0^{A_i}$ is the initial state; and $Q_m^{A_i} \subseteq Q^{A_i}$ is the set of marker states. Performing synchronous product of all automata in A $= \{A_i \mid i \in \mathrm{I}\}$, it results on automaton A where the set of states represents all possible combinations of activities being performed over a certain process instance.

Consider automaton A representing the uncoordinated behaviour of a set of activities A $= \{a_i \mid i \in \mathrm{I}\}$ of a process. Language L(A) represents all sequences of events that may be performed by these activities without any constraint, and L_m(A) is a subset of L(A) representing accomplished activities. The basic premise is that a process contains sequences of events in L(A) that are not acceptable because they violate some constraint. It is also current execution of activities. These sequences and states must be avoided. Also, it is possible that a sequence of events in $L_m(A)$ does not correspond to an accomplished task when the process instance is performed under supervision. Thus, that sequence needs to be unmarked by supervisor. Consider automaton S/A, such that $L_m(S/A) = supC(C/A)$, the one that recognizes the supremal controllable language of constraint activities.

Ramadge e Wonham (1989) define a supported traces as a sequence of events complying with all mandatory constrains. This definition complies with the definition of a sequence of events belonging to the supremal controllable language supC(C/A).

Figure 15 shows the languages relation: the region 1 includes sequences of events belonging to $supC(C/A) = L_m(S/A)$ (or supported traces); the region 2 includes sequences w such that $w \in L(A)$, $w \in Lm(A)$, $w \notin L(S/A)$, $w \notin Lm(S/A)$; the region 3 includes sequences $w \in L(A)$, $w \notin Lm(A)$, $w \in L(S/A)$, $w \notin Lm(S/A)$, the region 4 includes sequences $w \in L(A)$, $w \in Lm(A)$, $w \in L(S/A)$, $w \notin Lm(S/A)$.

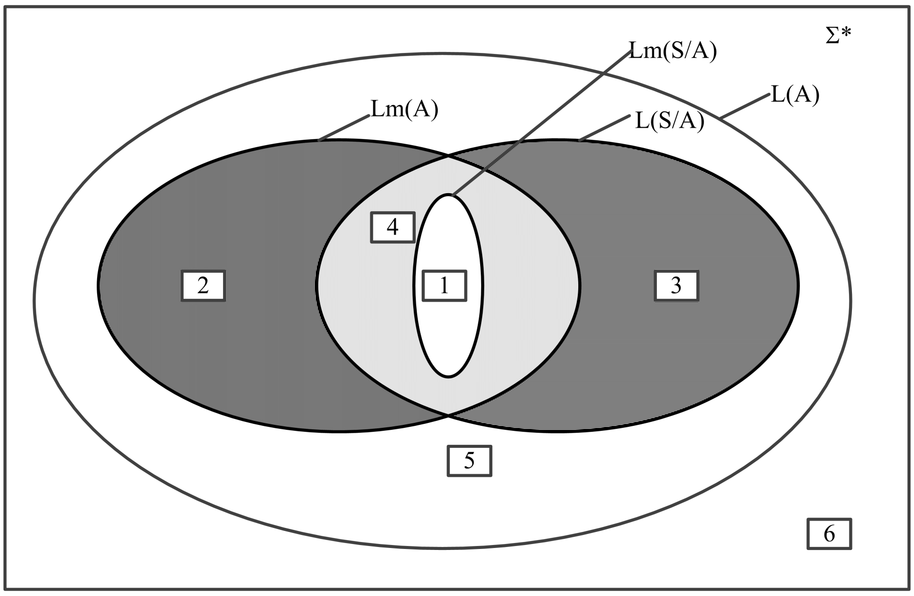


Figura 15 – Venn diagram of languages relation.

According to SCT, one step to formally obtain the supremal controllable language it is to express constraints in terms of automata. Usually, each constraint is represented as an automaton resulting in set $\{Cj \mid j \in J\}$, where J is a set of index uniquely identifying each constraint. Using the algorithms provided by SCT, the supremal controllable language represents that all related constraints will never be violated and there will always be at least one sequence of events leading to a marked state, i.e., there will always be the possibility of accomplishing a task.

The proposed approach is founded on the Supervisory Control Theory (SCT). The SCT is used as formalism to represent activities and constraints, so both modeled by automata.

In or approach, the constraints ensures that each process instance is performed under specific ordering of activities. Thus, depending on the selected constraints, it is possible to select a process variant. Also, each process variant correspond to the supremal controllable language obtained from SCT (and using specific constraints and activities automata models). The reasoning of this procedure is the core of our approach.

We believe that the Supervisory Control Theory (SCT) is a promising candidate to formalize the selection of process variant from a reference process model. We highlight the following reasons:

- SCT uses automata as formalism to represent activities, constraints and the resulting supremal controllable language. This is a formal and explicit way of representing them;
- Using SCT, from modelling to synthesis and visualization, the formal notation is used associated to a BPMN process model. Thus, at the same time we formalize the procedure of process variants selections and use a very well-known process notation;
- In SCT the state of each activity as well as each constraint may be easily visualized and understood by the end user at design-time;
- SCT provides algorithms to perform a formal synthesis of the supremal language instead of the usual manual and heuristic procedures;
- The obtained solution is minimally restrained and also dead-lock free. It means each selected process variant is correct by construction;
- New process variants may be rapidly and formally designed when modifications, such as redefinition of constraints or activities arrangements, are necessary;
- The set of possible process variants can be made to behave optimally with respect to a variety of criteria, where optimal means in minimally restrictive way (concern to the admissible language of the process). This is a very strong characteristic of the proposed approach. As far as we know, there is no approach that offer a better solution related to the admissible language.

3.4 Running example

We use an adapted version the healthcare process for handling examination presented in (REICHERT; WEBER, 2012a) and (AYORA et al., 2012) as running example to explain our approach. This process covers a family of process variants for handling examinations, including order handling, scheduling, transportation, and reporting. Figure 16 illustrated the reference process model. The grey rectangle represents a point where we specify which tasks have to be included (and the possible sequences). The tasks in the white rectangles represents common tasks that will appear in all variants.

We consider that a reference process model has two groups of tasks: (1) one group including tasks that always are in the process model (therefore common to all variants) , and (2) other one including tasks that are configurable (they may be in the model or not). In fact, the second group corresponds to the parts being subject to variation, which are commonly known as variation points (AYORA et al., 2012). Notice that in Figure 16 the first group includes the tasks in white rectangles and the second includes the tasks in the grey rectangle.

3.5 The proposed approach

This section presents the proposed approach that allows specialists of different fields of knowledge to represent a reference process model and select variants of it. We employ an adapted version of a health care process for handling medical examination which is presented in (REICHERT; WEBER, 2012a) and (HOMAYOUNFAR, 2012) to illustrated key concepts as well as this approach‘s application .

We consider that a reference process model is composed of a set of actions (A) such that there is an ordering among then. For instance, the reference process model presented at Figure 16 has 14 actions (A = a_0, a_1, ..., a_{14}), it specifies that action "a_3 - Order medical examination"may be performed only after action "a_1 - Request emergency medical examination"or "a_2 - Request standard medical examination"have been completed. It also specifies that actions "a_7 - Prepare patient"and "a_8 - Inform patient"may be performed concurrently. This figure represents a reference process model employing Business Process Modelling Notation (BPMN) (BPMN, 2011) which is a common notation employed in the business management field.

We propose that, IT' s and process' specialists, acting together represent a reference process model employing this notation. This notation, despite of being a formal way of representing a business process is quite intuitive, so it is appropriated to serve as a communication interface between two body of knowledge, the IT's world and the process' domain. An obtained BPMN graph formally describes the process specialist's knowledge.

Configurable connectors are included between actions whenever there is a choice about performing such actions. They specify logical relations, such as inclusive OR and exclusive OR, among actions succeeding it. For instance "Configurable connector 1"preceding actions "a_0 - Check-in for appointed medical examinationa_1 - Request emergency medical examination"and "a_2 - Request standard medical examination"represents an "exclusive OR"relation among them. In this case choosing to perform one action implies that the other actions cannot be performed. Configurable connectors 2 and 4 have a similar meaning.

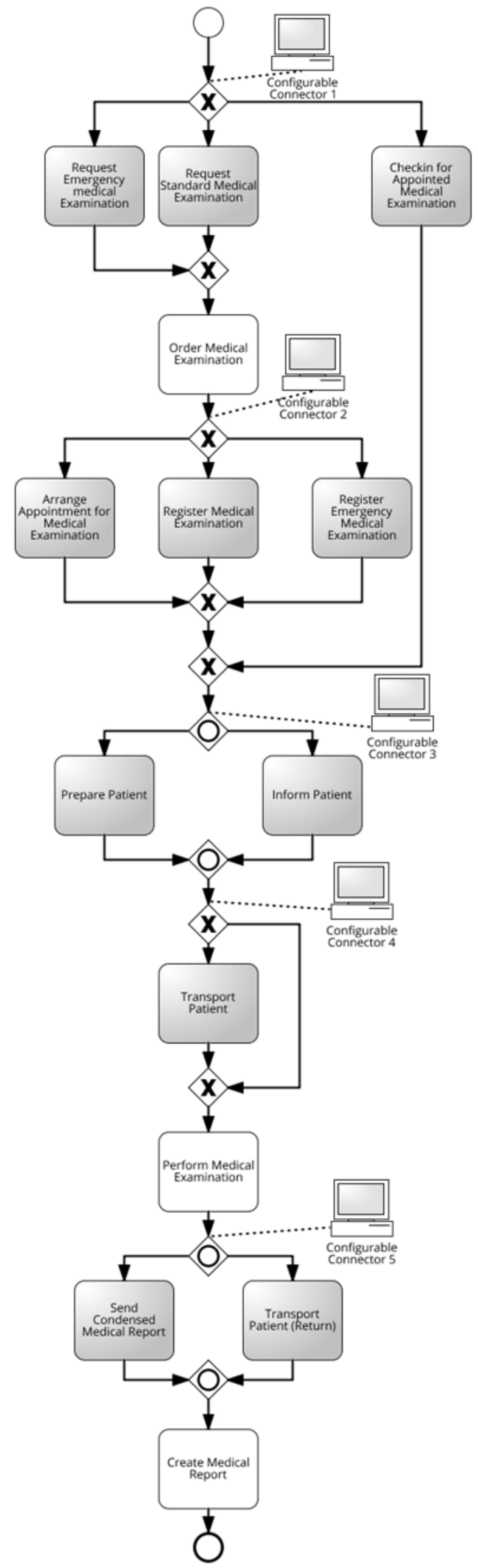

Configurable Connector 1
Request Emergency medical Examination
Request Standard Medical Examination
Checkin for Appointed Medical Examination
Order Medical Examination
Configurable Connector 2
Arrange Appointment for Medical Examination
Register Medical Examination
Register Emergency Medical Examination
Configurable Connector 3
Prepare Patient
Inform Patient
Configurable Connector 4
Transport Patient
Perform Medical Examination
Configurable Connector 5
Send Condensed Medical Report
Transport Patient (Return)
Create Medical Report

Figura 16 – Reference Process model adapted from (REICHERT; WEBER, 2012a).

Configurable connectors 3 and 5 represent an "inclusive OR"relation. From configurable connector 5 it is possible to choose to perform action "a_7 - Prepare patient"and do not perform action "a_8 - Inform patient", or to perform action a_8 and do not perform action a_7, or even, to perform both actions a_7 and a_8. In the last case, actions a_7 and a_8 may be performed in a strict sequence as, start and complete action a_7 them start and complete action a8, or they may be performed concurrently, such as, start action a_7 them start action a_8, complete action a_8 and finally complete action a_7. A configurable connector must be associated with closing connectors. In this figure, there are two closing connectors associated with "Configurable connector 1", one following actions a_1 and a_2 and another one following action a_0. Such interconnection represents that, if any of the first two actions (a_1 or a_2) is performed then they must be followed by action a_3 and by some of the other three actions succeeding "Configurable connector 2". On the other hand, if action a_0 is performed then none of the mentioned actions may be performed.

A reference process model includes all possible actions, and a process variant of it will include only a subset of it. The set of actions may be partitioned into a subset of mandatory actions (A_m) and a subset of optional actions (A_o), A mandatory action will be present at every variant while an optional action may or may not be present in a certain variant. In Figure 16, mandatory actions are represented by white backgrounded boxes and optional actions by grey backgrounded boxes. Actions "a_{10} - Perform medical examination"and "a_{13} - Create medical report"are the only mandatory ones in this reference process model.

In this approach, each action is a discrete event system, such that it may be represented as an automaton. A reference process model as well as each variant of it is a composed system. They are constituted by several subsystem (actions) that may be performed sequentially and/or concurrently.

Each action must be represented by an automaton (which is a labeled transition system). This is a common formalism employed in the computer science field as well as by the SCT. Figure 17 illustrates some, among several, possibilities of representing an action employing different levels of abstraction. Choosing one among them is based on the expected process behavior. Automata in Figure 17-a and 17-b are the best choices whenever it is considered that there never is the possibility of performing two or more actions concurrently and it is not considered the possibility of pausing and resuming a corresponding action. This is due to their simplicity as well as the resulting computational complexity involved in performing operations on automata. These automata express that an action is executed instantaneously. In automaton at Figure 17-a, initial state (circle with an incoming arrow) numbered 0 means that a corresponding action is idle. State 1 means that action has been performed. Arrow from state 0 to state 1 express that occurrence of event ei results in a state transition (from 0 to 1). This event means instantly execute action ai. Automata in Figure 17-c through 17-e allows representing concurrent execution of actions. State 1 in these automata means that a corresponding action is being performed. They employ events s_i (start executing action) and c_i (complete action).

Automata in Figures 17-a, 17-c and 17-e specify that the corresponding action may be performed only once in the process instance, while automata in Figures 17-b and 17-d specify that the corresponding action may be performed several times in the same process instance. In automaton at Figure 17-d it is possible to perform a sequence of events as "s_i c_i s_i c_i s_i c_i $\ldots$"while in automaton in Figure 17-c it is not possible to perform neither s_i neither c_i after the sequence of events "s_i c_i"has been performed. The set of states in an automaton is partitioned into marked states (double lined circle) and unmarked (single lined circle). Unmarked states express that something is pending to be accomplished. Marked states express that there is nothing pending, i.e., a task has been accomplished. Automata in Figure 17-c 17-d and 17-e express that once an action has been started (event s_i happened) it must be completed (event c_i needs to happen) to consider an accomplished task. This is because state 1 (Figure 17-c and 17-d) and states 1 and 2 (Figure 17-e) are not marked. Initial state in all automata are marked states, this means that even if the action is not performed, a task has been accomplished. They may be employed to represent optional actions. Mandatory actions must be represented with an unmarked initial state.

The SCT introduces the concept of event controllability. An event is considered controllable if a control agent (a supervisor under SCT) may avoid or disable its occurrence, it is considered uncontrollable if its occurrence cannot be disabled. Specifying event controllability is a modelling decision and influences the resulting system's behaviour. In a process variant only a subset of actions must be executed, so it is necessary to avoid start executing remaining actions. Due to this, event start executing action must be a controllable event. If it is not possible to avoid completing an action once it is being performed then event complete action is uncontrollable. Different automata models may be employed to represent actions of a certain reference process model. In this application example all actions have been represented by automata with the structure as in Figure 17-c.

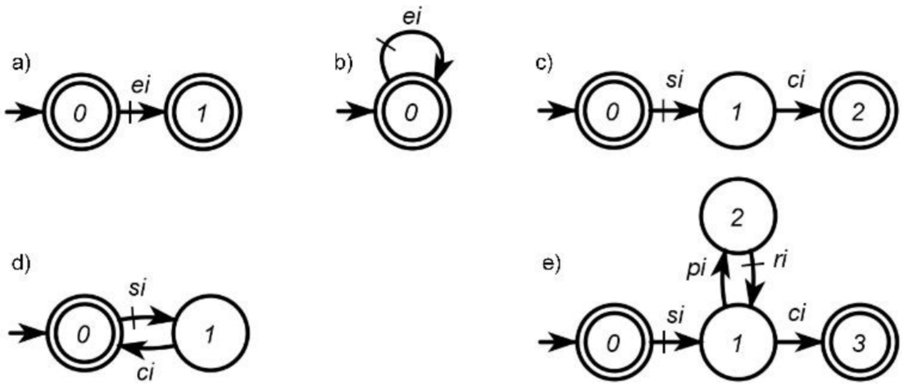
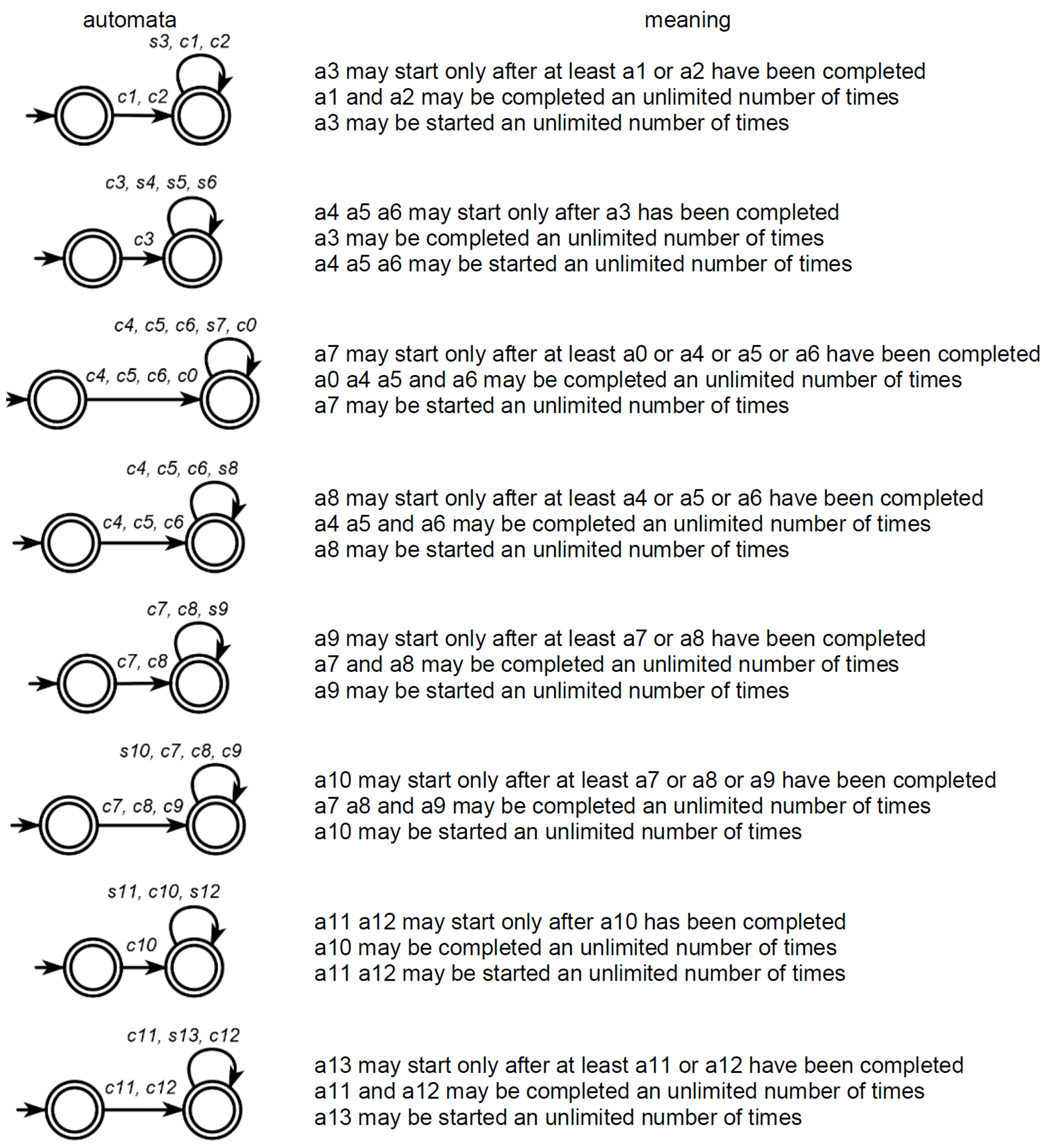


Figura 17 – Possible automata representing an action

A process' specialist textually specifies ordering among actions as well as maximum number of times a certain task may be performed in a reference process model. These constraints are represented in a BPMN graph and translated into automata by an IT's specialist. Events in these automata must be only those employed to represent actions. Figure 18 presents automata employed to express constraints in the application example and corresponding meaning.



a3 may start only after at least a1 or a2 have been completed
a1 and a2 may be completed an unlimited number of times
a3 may be started an unlimited number of times

a4 a5 a6 may start only after a3 has been completed
a3 may be completed an unlimited number of times
a4 a5 a6 may be started an unlimited number of times

a7 may start only after at least a0 or a4 or a5 or a6 have been completed
a0 a4 a5 and a6 may be completed an unlimited number of times
a7 may be started an unlimited number of times

a8 may start only after at least a4 or a5 or a6 have been completed
a4 a5 and a6 may be completed an unlimited number of times
a8 may be started an unlimited number of times

a9 may start only after at least a7 or a8 have been completed
a7 and a8 may be completed an unlimited number of times
a9 may be started an unlimited number of times

a10 may start only after at least a7 or a8 or a9 have been completed
a7 a8 and a9 may be completed an unlimited number of times
a10 may be started an unlimited number of times

a11 a12 may start only after a10 has been completed
a10 may be completed an unlimited number of times
a11 a12 may be started an unlimited number of times

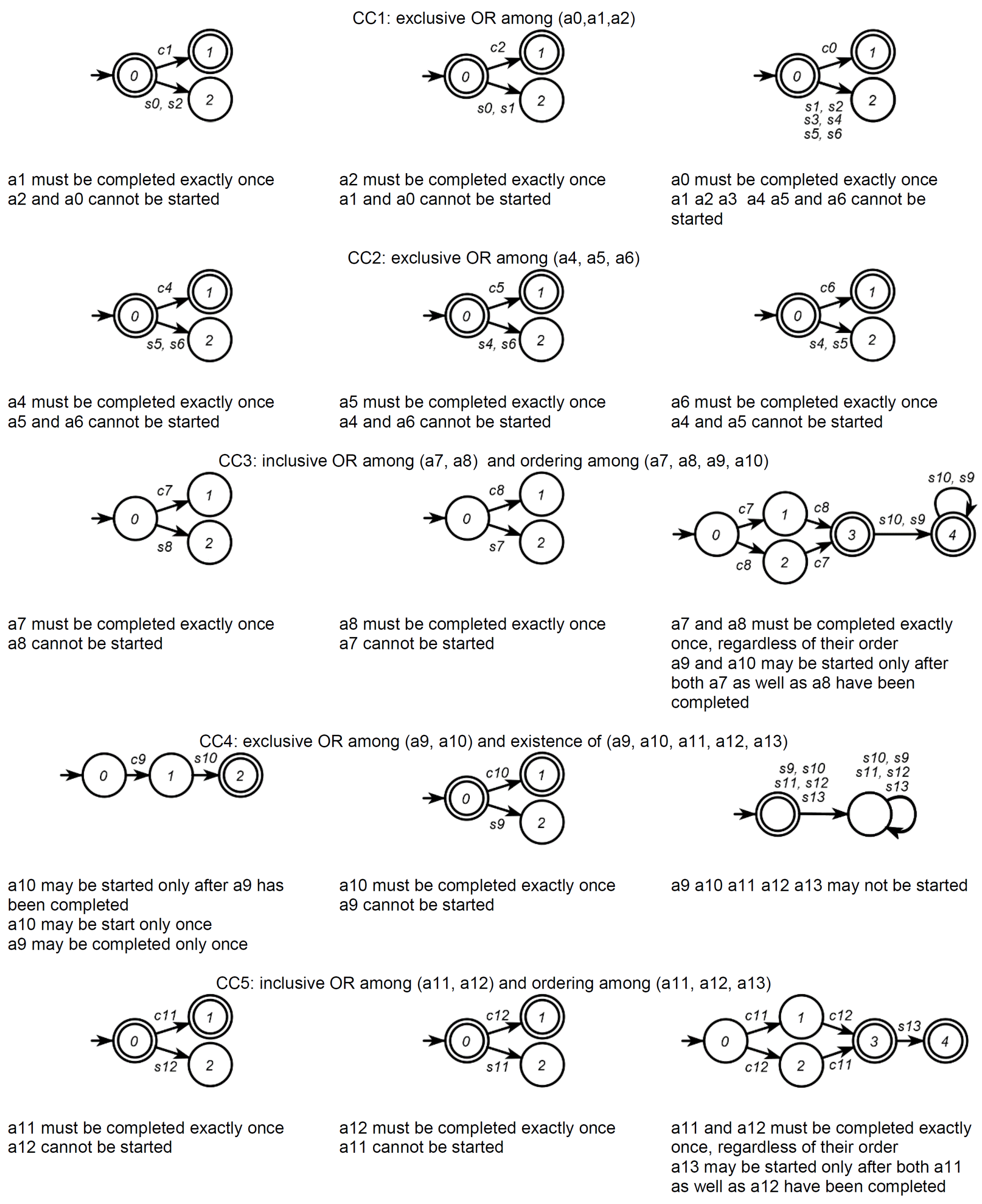
a13 may start only after at least a11 or a12 have been completed
a11 and a12 may be completed an unlimited number of times
a13 may be started an unlimited number of times

Figura 18 – Automata representing constraints on ordering of actions

Performing the operation named synchronous product (CASSANDRAS; LAFORTUNE, 2009) of all automata representing actions results an automaton representing every possibility of performing actions without any constraint. Under the SCT it represents a system's uncoordinated behaviour, usually represented as G. Performing the synchronous product of automaton G with all automata representing constraints results an automaton representing a constrained behaviour of G, usually represented as E. This behaviour may result uncontrollable and/or blocking. Automaton E is uncontrollable if, in order to satisfy the set of constraints, it is required to prohibit the execution of an uncontrollable event. It is blocking if reaching an unmarked state there is no possible event to be performed leading to a marked state. It is then necessary to obtain the supremal controllable language (RAMADGE; WONHAM, 1989; RAMADGE; WONHAM, 1987) of E regarding G, which is represented by automaton RPM = SupC(E,G), this represents a reference process model. These operations on automata are performed employing software made available by the SCT's scientific community. Examples of software are XPTCT (WONHAM, 2013) UMDES (RICKER; LAFORTUNE; GENC, 2006).

A total of 14 action automata and 8 constraint automata where employed in this application example. The automaton representing the reference process model has a total of 945 (DERGUECH; VULCU; BHIRI, 2010). An indexing structure for maintaining configurable process models. This figure allows one to compare a BPMN graph and a corresponding automaton. While the first one is quite intuitive and compact, it omits details about action's behaviour and constraints among them. An automaton explicitly represents all possible sequences of events and thus may have a huge number of states making impossible to one to completely visualize it. On the other hand, it allows one to develop an information system aiming to manage and to supervise the execution of process variants instances. In this proposed approach, translating textual language into automata as well as performing operations on automata may be left for an IT's specialist avoiding to burden a process specialist.

A process variant is obtained by imposing further constraints on the reference process model specifying (non)existence of actions as well as refining ordering of actions and number of times of performing actions. Such constraints cannot contradict or be less restrictive than those already imposed to the reference process model, they can only be more restrictive. As before, a configurable connector represents logical relations among actions succeeding it. These relations lead to constraints to be imposed on a reference process model. Choosing one constraint at each configurable connector results a process variant. We propose that, while representing a BPMN graph representing a reference process model, IT's and process' specialists, textually formulate the meaning of constraints associated with configurable connectors. These will be translated into automata by an IT's specialist. Figure 19 presents the meaning of configurable connectors, associated constraints and corresponding automata. For instance, in automaton $c3_1$ there is an arrow from state 0 (initial state) to state 1 labeled with event s_8. Since state 1 is unmarked, this event will be exclude from a process variant when performing those operations on automata. It will avoid action a_8 to be started. Since a marked state (state 2) is reached only after occurrence of event c_7 then an accomplished task is not recognized until completing action a_7. In a similar way in automaton $c3_3$, an accomplished task is not recognized until completing both actions a_7 and a_8, doesn't matter their relative order. Since events s_9 and s_{10} appear only after the occurrence of both c_7 and c_8 then actions a_9 and a_{10} may start only upon completing a_7 as well as a_8.



CC1: exclusive OR among (a0,a1,a2)

a1 must be completed exactly once
a2 and a0 cannot be started

a2 must be completed exactly once
a1 and a0 cannot be started

a0 must be completed exactly once
a1 a2 a3 a4 a5 and a6 cannot be started

CC2: exclusive OR among (a4, a5, a6)

a4 must be completed exactly once
a5 and a6 cannot be started

a5 must be completed exactly once
a4 and a6 cannot be started

a6 must be completed exactly once
a4 and a5 cannot be started

CC3: inclusive OR among (a7, a8) and ordering among (a7, a8, a9, a10)

a7 must be completed exactly once
a8 cannot be started

a8 must be completed exactly once
a7 cannot be started

a7 and a8 must be completed exactly once, regardless of their order
a9 and a10 may be started only after both a7 as well as a8 have been completed

CC4: exclusive OR among (a9, a10) and existence of (a9, a10, a11, a12, a13)

a10 may be started only after a9 has been completed
a10 may be start only once
a9 may be completed only once

a10 must be completed exactly once
a9 cannot be started

a9 a10 a11 a12 a13 may not be started

CC5: inclusive OR among (a11, a12) and ordering among (a11, a12, a13)

a11 must be completed exactly once
a12 cannot be started

a12 must be completed exactly once
a11 cannot be started

a11 and a12 must be completed exactly once, regardless of their order
a13 may be started only after both a11 as well as a12 have been completed

Figura 19 – Constraints and automata associated with configurable connectors

In order to help properly selecting constraints leading to process variants, IT's and process' specialists may formulate a questionnaire associating configurable connector's with questions. Each question may have several answering choices. Choosing exactly one choice selects a subset of constraints to be further imposed on the reference process model. It is possible that, in order to obtain coherent process variants, choices in different questions become mutually exclusive. Correlation among choices must be specified during questionnaire formulation. Due to these correlations it is possible that the number of formulated questions results smaller than the number of configurable connectors. It is also possible that, due to a selected choice in a question, a further question results meaningless and may be hidden from a responder. Table 3 presents the guided questionnaire associated with the BPMN graph on Figure 16. Due to coherent process variants, it requires only two questions. It can be seen that constraint $c5_2$ is never selected.

Tabela 3 – Guided questionnaire

Question	Correlation among choices and questions x ⊕ y : choice x and choice y are mutually exclusive. x [Z] : choice x hides question Z.	Subset of constraints selected by choice x
Q1) What kind of attendance is to be performed?		
1.1 - Emergency medical examination	1.1 ⊕ 2.1	{ c1_1 , c2_3, c3_1 , c4_1, c5_3 }
1.2 - Appoint medical examination	1.1 ⊕ 2.2	{ c1_2 }
1.3 - Check-in for appointed medical examination	1.1 [Q2]	{ c1_3, c3_1 , c4_2 , c5_1 }
	1.3 ⊕ 2.1	
	1.3 ⊕ 2.2	
	1.3 [Q2]	
Q2) Is immediate attendance available?		
2.1 - No		2.1 { c2_1 , c3_2 , c4_3 }
2.2 - Yes		2.2 { c2_2 , c3_3, c4_2, c5_1 }

Answering a questionnaire selects a subset of constraints. Performing the synchronous product of automaton RPM with all selected automata results an automaton representing a constrained behavior of RPM, it may be represented by E$_{PV}$. Again, this behavior may result uncontrollable and/or blocking. It is necessary to obtain the supremal controllable language of E_{PV} regarding RPM, which is represented by automaton PV = SupC(E$_{PV}$, RPM). This automaton represents the behavior of the corresponding process variant. Different answers on the questionnaire will select different subsets of constraints thus resulting different process variants.

There are four coherent process variants associated with reference process on Figure 16. Selecting choice 1.1 or 1.3 at question Q$_1$ selects all necessary constraints to be imposed on the reference process model in order to obtain two of its variants, so question Q$_2$ may be hidden from a responder. Selecting choice 1.2 at question Q$_1$ selects a single constraint and it is necessary to answer question Q$_2$. The constraints to be imposed in order to obtain the other two variants are selected at question Q$_2$. Without a guided questionnaire one would be able to select a combination of constraints leading to an uncoherent process variant, for instance selecting $c1_1$ (request emergency medical examination) together with $c2_1$ (arrange appointment for medical examination). Figure 20 presents automaton representing a process variant obtained selecting "1.2 - Appoint medical examination"at question Q_1 and "2.2 - Yes"at question Q_2. It also presents a corresponding BPMN graph.

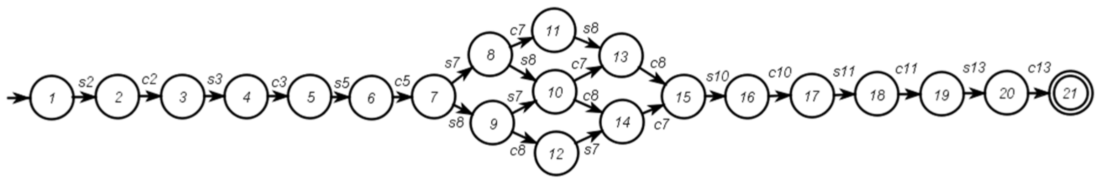


Figura 20 – Process variant representing an emergency medical examination

3.6 Conclusion

This paper presented a formal approach for dealing with variability in BPM. The approach relies on a questionnaire composed of questions and answers associated to a set of constraints. To do this, we propose a formal method to connect the questionnaire with the process models variants. We do that associating each answers to a set of tasks that have to exist, and a set of tasks that have to not exist in a certain process. Then we define a supervisor that restrains the behavior of the reference process model according to existence and non-existence constraints types, thus deriving an individualized version of such reference process. The development of the approach has been motivated by the need to support the configuration of business process models variants, and our contribution is to provide a formal procedure to do it.

We point out that it would be possible to develop a software package that guides the process of questionnaire answering, signaling that a certain choice is not allowed to be selected or automatically selecting choices based on previously selected choices. Upon completing a questionnaire's answering, it may also automatically perform operations on automata in order to synthesize the automaton representing a process variant. However, it is out of the scope of this paper to discuss this implementation.

A BPMN graph, textual meaning actions' behaviour, textual meaning of constraints on configurable connectors, and a questionnaire, will formally document requirements specifications of a project involving a process' specialist and an IT' specialist employing a common language to both. Corresponding automata will allow an IT's specialist to formally synthesize a reference process model and its variants employing SCT. Thus, we believe that both specialists executing a project using a common language will benefit both and the whole project process.

4 Simple Declarative Language (SDL): a conceptual framework to model constraint based processes

Abstract

Constraint-based processes have received increased interest for featuring non-standardized settings. In these processes, the control flow is defined implicitly as a set of constraints or rules, and any possibility that is not in violation of any of the constraints set is allowed to be executed. A constraint-based process model specifies the tasks that must be performed to produce the expected results but does not establish exactly how these tasks should be performed, i.e. any tasks can be performed provided the constraints are not violated, with user preferences driving process execution. Constraint based processes are better modeled by declarative languages. Declare and DCR graphs are examples of frameworks applied in modelling constraint based models by using declarative languages. The SCT approach for modeling constraint based processes is an example of formalism intended to model constraint based models by using declarative language. This paper presents some of the main features of the Declare, DCR graphs and SCT approaches. This paper's main contribution is introduced after presenting the Declare, DCR graphs and SCT approaches for modeling constraint based processes: the Simple Declarative Language (SDL) framework, a new conceptual framework for modeling constraint based processes.

Keywords: Constraint based processes, Declare / Linear Temporal Logic, Dynamic Condition Response graphs, Supervisory Control Theory, soundness.

4.1 Introduction

There has been an increased interest in constraint-based processes because of having non-standardized settings (REIJERS; SLAATS; STAHL, 2013). Non-standardized settings provide users the power to choose tasks, procedures and methods to be executed in a given process (MERTENS; GAILLY; POELS, 2015b). In general, users make these choices according to their professional expertise and the context in which the process is being performed (UNGER; LEOPOLD; MENDLING, 2015). Another important characteristic of constraint-based processes is that they permit easy identification of business rules or constraints with which the process must be compliant (GOEDERTIER; VANTHIENEN;

CARON, 2015).

Declarative languages are better suited for modeling constraint-based processes because they facilitate the formal declaration of constraints or business rules (HAISJACKL et al., 2013). These languages define the tasks that must be performed to produce the expected results but not establish exactly how these activities should be performed (MERTENS; GAILLY; POELS, 2015a).

Thus, despite process execution being user-choice driven, users can only choose task sequences that do not violate any process constraints (SLAATS et al., 2013). In constraint-based processes, set of constraints defines implicitly the control flow (LY et al., 2015). Constraint based process are modeled by environments that provide some kind of support tool for users to model and run the process. Users model the process when designing it. The time when users run the process is known as the process run time.

These environments that provide some kind of support tool to model and run processes are called constraint based process modeling frameworks. Figure 21 shows a very simplified process design and run framework.

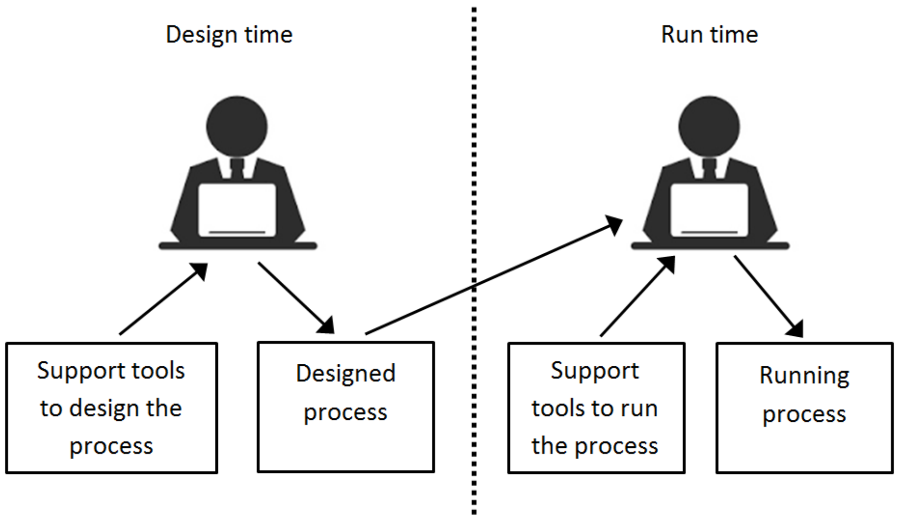

Design time
Run time
Support tools to design the process
Designed process
Support tools to run the process
Running process

Figura 21 – Very simplified framework to design and run a process

There are at least two important frameworks intended to model constraint based process: Declare (PESIC, 2008; MONTALI et al., 2013; CICCIO et al., 2015), and DCR graphs (MUKKAMALA, 2012).

Declare is a framework that provides templates of tasks and constraints for modeling and performing constraint-based processes, each template is defined by a Linear Temporal Logic (LTL) expression (MAGGI, 2013; AALST; PESIC; SCHONENBERG, 2009; PEŠIĆ; BOŠNAČKI; AALST, 2010; PESIC, 2008; MONTALI et al., 2013).

Dynamic Condition Response (DCR) graphs is a framework that provides sets of events and relations for modeling and performing constraint-based processes, each task can be represented as an event (MUKKAMALA, 2012; MUKKAMALA; HILDEBRANDT; SLAATS, 2013; HILDEBRANDT et al., 2013; DEBOIS; HILDEBRANDT; SLAATS, 2015; ESHUIS et al., 2016; DEBOIS et al., 2016).

The Supervisory Control Theory (SCT) is mathematical formalism for synthesis of optimal discrete event systems (DES) controllers (RAMADGE; WONHAM, 1987). SCT presumes that a set of tasks may display uncontrollable behaviors that might violate some of the required properties (WONHAM; RAMADGE, 1987). This behavior must be modified through an agent, the supervisor, in order to deliver a set of specifications or to ensure that certain restrictions are not violated (RAMADGE; WONHAM, 1989). The SCT approach for modeling constraint based processes (SANTOS et al., 2014) is based on the Supervisory Control Theory and provides a set of automata to represent business process task and constraint modeling (SANTOS et al., 2011; SANTOS et al., 2014; SCHAIDT et al., 2013; CESTARI et al., 2014; SCHAIDT et al., 2013).

There are several important concepts that are common to constraint based processes. This paper deals with four of them: soundness (AALST et al., 2011), enabled events (HILDEBRANDT et al., 2013), pending events (MUKKAMALA, 2012) and violation of constraints (PESIC; SCHONENBERG; AALST, 2007).

Soundness can be described as the combination of three behaviors in a process model: *option to complete*, *proper completion* and *no dead activity* (AALST et al., 2011). When a process fulfills the *option to complete*, this means that the process will always reach completion. When a process fulfills *proper completion*, this means that whenever the process is completed, all its tasks are completed and no tasks are left running. When a process fulfills *no dead task*, this means that every task in the process may be completed at least once (AALST, 2015; MONTALI; CALVANESE, 2016; AALST et al., 2011).

The concept of soundness is important in semantic process analysis (AALST et al., 2011; FAVRE; FAHLAND; VÖLZER, 2015). For example, if a process violates *option to complete* there is a sequence of events in this process that will lead to a deadlock, i.e. a state in which it is impossible to finish the process properly. Declare (MAGGI et al., 2011; MAGGI et al., 2011; MAGGI; MOOIJ; AALST, 2011; PEŠIĆ; BOŠNAČKI; AALST, 2010) and DCR (MUKKAMALA, 2012; HILDEBRANDT et al., 2013; HILDEBRANDT; MUKKAMALA, 2011) are frameworks that offer tools enabled to perform process analysis by checking for violation of soundness requirements. These approaches identify and inform the requirements that have been violated.

An event is enabled at a state of the process if this event can be executed in this state of the process. Declare (MAGGI et al., 2011; MAGGI et al., 2011; MAGGI; MOOIJ; AALST, 2011; PEŠIĆ; BOŠNAČKI; AALST, 2010) and DCR (MUKKAMALA, 2012; HILDEBRANDT et al., 2013; HILDEBRANDT; MUKKAMALA, 2011) are frameworks that provide support for enabled events.

An event is pending in a state of the process if executing this event is mandatory in this state of the process. DCR framework (MUKKAMALA, 2012; HILDEBRANDT et al., 2013; HILDEBRANDT; MUKKAMALA, 2011) informs users the pending events in the process.

A constraint is a rule of the process, if this rule is not obeyed, then it has been violated. Declare frameworks (MAGGI et al., 2011; MAGGI et al., 2011; MAGGI; MOOIJ; AALST, 2011; PEŠIĆ; BOŠNAČKI; AALST, 2010) inform users the constraints that have been violated in the process.

This paper presents how the Declare and DCR graphs frameworks deal with *soundness*, enabled events, pending events and constraint violations. The main grounds for the SCT constraint based process modeling approach are also introduced.

The main contribution of this paper is introduced after the Declare, DCR graphs and SCT approach for modeling constraint based processes are presented: the Simple Declarative Language (SDL) framework. The Simple Declarative Language (SDL) framework is a new conceptual framework for modeling constraint based processes. The SDL framework provides models of tasks and constraints to be deployed in modeling constraint based processes. Tasks and constraints in SDL frameworks are based on SCT approach models. The SDL framework offers a set of three constraints only. At design time, the SDL framework provides support in designing soundness compliant constraint based processes. At run time, the SDL framework provides support to enabled and pending events.

This paper is divided into five sections. Section 2 provides the foundations for *soundness*. The imperative Workflow Net language is used to conceptually present soundness. Section 3 provides the foundations of Declare. Section 4 provides the foundations of DCR graphs. Section 5 provides the foundations of SCT approach. Section 6 provides foundations of SDL frameworks. Section 7 sets out the conclusions.

4.2 Soundness

Soundness can be described as the combination of three behaviors in a process: *option to complete*, *proper completion* and *no dead activity* (AALST, 2015; MONTALI; CALVANESE, 2016; AALST et al., 2011). When a process fulfills the *option to complete*, this means that the process will always reach completion. When a process fulfills *proper completion*, this means that whenever the process is completed, all its tasks are completed and no tasks are left running. When a process fulfills *no dead task*, this means that every task in the process may be completed at least once (AALST, 2015; MONTALI; CALVANESE, 2016; AALST et al., 2011).

There are studies about methods to verify soundness in business processes (CLEMPNER, 2014a; CLEMPNER, 2014b; ESPARZA; HOFFMANN, 2016; KHERBOUCHE; AHMAD; BASSON, 2013; AALST; HIRNSCHALL; VERBEEK, 2002; LIU; JIANG, 2015; LIU et al., 2014). The concepts of soundness are often used in business processes generated by imperative languages. In fact, these concepts were initially defined for this class of processes.

Business processes generated by imperative languages are called imperative, rigid or highly structured processes. Imperative languages are better suited for modeling these processes because, unlike declarative languages, they define exactly how a set of tasks should be performed. Thus, a model that explicitly defines the order and execution of activities is required. Examples of imperative languages are Business Process Model and Notation (BPMN) (CHINOSI; TROMBETTA, 2012) and Event-driven Process Chain (EPC) (MENDLING; NEUMANN; NÜTTGENS, 2015), among others.

So before presenting three examples of soundness violations, we will introduce the basic concepts of Workflow Net (AALST, 2000; SALIMIFARD; WRIGHT, 2001; GIRAULT; VALK, 2013). WorkFlow Net is a particular type of Petri Nets (MURATA, 1989) that can be used to model imperative business processes (AALST, 2000; SALIMIFARD; WRIGHT, 2001; GIRAULT; VALK, 2013). WorkFlow Net is the imperative language used in this section to present examples of soundness violations (FLENDER; FREYTAG, 2006; FAHLAND et al., 2009a; AALST, 2000). Definitions of Petri Net and Workflow Net are presented next.

A process is syntactically correct when it fulfills the syntax rules of the language that models it. For a process represented by a workflow net, the definitions of Petri Net and Workflow Net must be known the since these definitions establish the syntax rules for any workflow net. Below, the definitions of Petri Net (MURATA, 1989) and Workflow Net (AALST, 2000) are presented.

Definition 4.2.1. *A Petri net is a triple PN = (P, T, F), such that:*

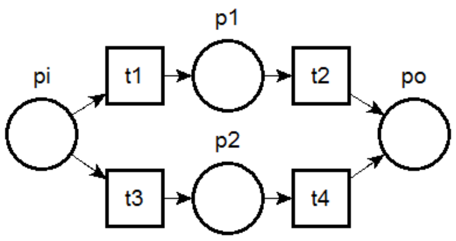
- *P is a finite set of places*
- *T is a finite set of transitions ($P \cap T = \emptyset$),*
- *$F \subseteq (P \; x \; T) \cup (T \; x \; P)$ is a set of arcs (flow relation).*

Definition 4.2.2. *Let PN = (P, T, F) be a Petri net and F* be the reflexive transitive closure of F. PN is a Workflow net (WF-net) iff:*

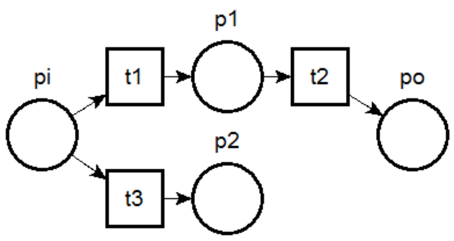
- *there exists exactly one input place: $\exists! p_I \in P$, $\bullet p_I = \emptyset$.*
- *there exists exactly one output place: $\exists! p_O \in P$, $p_O \bullet = \emptyset$.*
- *each node is on a directed path from the input place to the output place: $\forall n \in P \cup T[(p_I, n) \in F^* \wedge (n, p_O) \in F^*]$.*

In Definition 4.2.2, notations $\bullet p_I$ and $p_O \bullet$ represent, respectively, the pre-set of the input place and post-set of the output place. This notation is general, i.e. $\bullet p_i$ and $p_i \bullet$ represent respectively the pre-set of any place i and post-set of any place i. The pre-set of place i is composed by all transitions coming into place i. The post-set of place i is composed by all the transitions coming out from place i. In Definition 4.2.2, a node is a place or a transition.

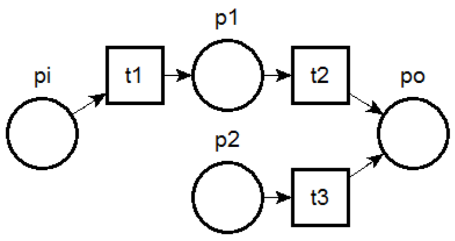
Definition 4.2.1 and 4.2.2 establish the syntax for a Petri net and that a workflow net is a Petri net with only a single input and a single output place, and, for every node, (place or transition) there is a path from the input place to the node and from the node to the output place. These conditions must be obeyed in order to fulfill the syntax of a workflow net. For example, Figure 22(a) presents a syntactically correct workflow net having only one input place and one output place, and for every place or transition that pertains to the process, there is at least one sequence of places and transitions from the input place p_I to this place or transition, and at least one sequence of places and transitions from this place or transition to the output place p_O. Figure 22(b) presents a non-syntactically correct workflow net in which there is no sequence of places and transitions from t_3 and p_3 to output place p_O. Figure 22(c) presents other non-syntactically correct workflow nets where there is no sequence of the places or transitions from input place p_i up to p_2 or t_3.



(a) A syntactically correct workflow net



(b) A non syntactically correct workflow net



(c) A non syntactically correct workflow net

Figura 22 – Example of syntactically correct and incorrect workflow nets

The definition of soundness workflow net (AALST, 2015; MONTALI; CALVANESE, 2016; AALST et al., 2011) is presented. A Workflow Net that complies with soundness is called a semantically correct Workflow Net (ROSA, 2009).

Definition 4.2.3. *The workflow net complies with soundness if it complies with three requirements:*

• *option to complete: the token in the initial place can always reach the output place, for any sequence of transitions firing from the initial place.*

• *proper completion: the token reaches the output place then there is not token at any other place.*

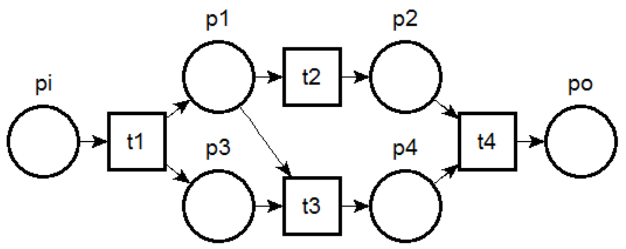
• *no dead activity: every transition can be enabled in order to fire the tokens in its incoming places.*

Figures 22(b) and 22(c) show two examples of syntax violation, but they are also two examples of soundness violation. The workflow net in Figure 22(b) is non-sound because sequence $p_i.t_3.p_2$ violates *option to complete.* The workflow net in Figure 22(c) is non-sound because transition t_3 is never enabled to fire its incoming tokens, so t_3 is a dead transition and the *no dead task* requirement is violated. Figures 22(b) and 22(c) demonstrate that if the syntax of a workflow net is violated, then the soundness can be also violated. However, it is not always true that if the syntax of a workflow net is not violated then the soundness is not violated. Syntax correctness is a required, but not sufficient condition, to guarantee that a workflow net is behaviorally sound. The other requirement that workflow nets must comply with, to be behaviorally sound, is *free-choice.* The definition of a free-choice workflow net is shown below (AALST, 1997).

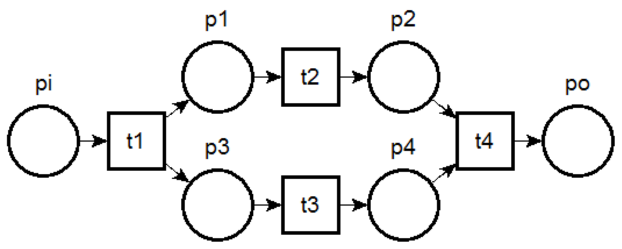
Definition 4.2.4. *Let N = (P, T, F) be a Petri net. N is free-choice (FC) if for every couple of places the transitions in their postset are exactly the same or totally different. Formally:*

$$\forall\ p_I, p_O \in P\ [p_1\bullet \bigcap p_2\bullet \neq \emptyset] \Rightarrow p_1\bullet = p_2\bullet].$$

For example, the process represented by the workflow net in Figure 3(a) is syntactically, but not semantically correct, this happens because this workflow net is not free-choice, i.e. $p_1\bullet \bigcap p_3\bullet = \{t_3\} \neq \emptyset$, $p_1\bullet = \{t_2, t_3\}$, $p_3\bullet = \{t_3\}$, $p_1\bullet \neq p_3\bullet$, violating the free-choice condition . The consequence of the workflow net in Figure 23(a) not being free-choice is presented next. If t_1 is executed, then t_2 and t_3 are enabled. If t_2 is fired before t_3, then t_3 is permanently disabled. If t_3 is fired before t_2, then t_2 is permanently disabled. As t_2 and t_3 are mutually exclusive, t_4 will never be fired and the process cannot be completed. Thus option to complete is violated and the process is not sound. Figures 23(b) shows the way to turn the process in Figure 23(a) into a behaviorally sound one. In Figure 23(b), the pathway from p_1 to t_3 was removed such that $p_1\bullet \cap p_3\bullet = \emptyset$.



(a) A syntactically correct but semantically incorrect workflow net



(b) A syntactically and semantically correct workflow net

Figura 23 – Example of semantically correct and incorrect workflow nets

4.3 Declare

Declare is a framework for modeling and performing constraint-based processes. This section presents the main grounds of the Declare framework and how it deals with soundness, enabled events and pending events.

4.3.1 Tasks and constraints in Declare

Declare provides templates of tasks and constraints. Each task is divided into three events: start (s), complete (c) and cancel (x). Figure 24 shows the transition system of a task in Declare, after a task is started (s is executed), it may be completed or canceled (c or x may be performed), if the task is completed (c is executed) then it was successfully executed, but if the task is canceled (x is executed) then its execution has failed (PESIC, 2008). Events s, c and x are indexed according to the tasks to which they pertain, so events s_i, c_i and x_i pertain to task t_i, events s_j, c_j and x_j pertain to the task t_j, and so on.

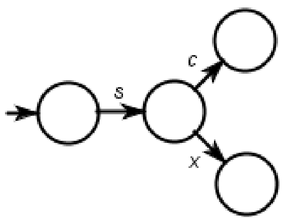


Figura 24 – Automaton that represents a task in Declare framework

Declare offers four sets of constraints: *existence*, *relation*, *choice* and *negation*. *Existence* models specify how often or when a task can be performed. *Relation* models define some relation between two (or more) tasks. *Negation* models define a negative relation between tasks. *Choice* models are used to specify that one must choose between two or more tasks. Examples of *existence* models are *existence*(t_i), *existence2*(t_i), *absence2*(t_i), *exactly1*(t_i). Examples of *relation* models are *responded existence*(t_i,t_j), *coexistence*(t_i,t_j), *response*(t_i,t_j), *precedence*(t_i,t_j). Example of *choice* model is *exclusive 1of2*(t_i,t_j). Examples of *negation* models are *not responded existence*(t_i,t_j) and *not coexistence*(t_i,t_j) (PESIC, 2008). Table 4 presents some constraints with their descriptions.

Tabela 4 – Constraints from Declare and their description

Name of the constraint	Description
existence(t_i)	the event c_i must be performed at least once
existence2(t_i)	the event c_i must be performed at least twice
absence2(t_i)	the event c_i must be performed at most once
exactly1(t_i)	the event c_i must be performed exactly once
responded existence(t_i, t_j)	if the event c_i is performed then the event c_j must be performed
coexistence(t_i, t_j)	if the event c_i is performed then the event c_j must be performed and vice versa
response(t_i, t_j)	every time the event c_i is performed the event c_j must be performed afterward
precedence(t_i, t_j)	event c_i must be performed one time before any instance of the event c_j be performed
exclusive 1of2(t_i, t_j)	the event c_i or exclusively the event c_j must be performed
not responded existence(t_i, t_j)	if the event c_i is performed then the event c_j must not be performed
not co existence(t_i, t_j)	if the event c_i is performed then the event c_j must not be performed and vice versa

In Declare, each constraint is defined by a Linear Temporal Logic (LTL) expression (KESTEN; PNUELI; RAVIV, 1998). There are five temporal operators defined in LTL: operator *always* ($G(p)$), operator *next* ($X(p)$), operator *eventually* ($F(p)$), operator *until* (pUq), operator *weak until* (pWq). Operator $G(p)$ defines that p has to hold true throughout the entire subsequent path. Operator $X(p)$ defines that p has to hold true at the next state. Operator $F(p)$ defines that p eventually has to hold true somewhere on the subsequent path. Operator pUq defines that p has to hold true at least until q, which holds true at the current or a future position. Operator pWq is similar to operator until (U), but it does not require that q ever become true (SISTLA; CLARKE, 1985; GERTH et al., 1996; MAGGI et al., 2011). Table 5 presents the constraints from the Table 4 with their LTL expressions.

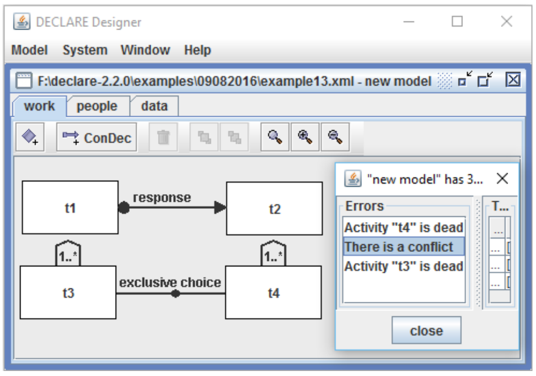
Tabela 5 – Constraints from Declare and their LTL expressions

Name of the constraint	LTL expression
$\mathit{existence}(t_i)$	$F(t_i,\, c_i)$
$\mathit{existence2}(t_i)$	$F((t_i,\, c_i) \wedge X(\mathit{existence}(t_i)))$
$\mathit{absence2}(t_i)$	$\neg \mathit{existence2}(t_i)$
$\mathit{exactly1}(t_i)$	$\mathit{existence}(t_i) \wedge \text{absence2}(t_i)$
$\mathit{responded\ existence}(t_i,\, t_j)$	$F(t_i,\, c_i) \Rightarrow F(t_j,\, c_j)$
$\mathit{coexistence}(t_i,\, t_j)$	$F(t_i,\, c_i) \Leftrightarrow F(t_j,\, c_j)$
$\mathit{response}(t_i,\, t_j)$	$G((t_i,\, c_i)\ ) \Rightarrow F(t_j,\, c_j))$
$\mathit{precedence}(t_i,\, t_j)$	$(\neg((t_j,\, s_j\ ) \vee (t_j,\, c_j) \vee (t_j,\, x_j)))W(t_i,\, c_i)$
$\mathit{exclusive\ 1of2}(t_i,\, t_j)$	$(F(t_i,\, c_i) \wedge \neg F(t_j,\, c_j)) \vee (\neg F(t_i,\, c_i) \wedge F(t_j,\, c_j))$
$\mathit{not\ responded\ existence}(t_i,\, t_j)$	$F(t_i,\, c_i)) \Rightarrow \neg(F(t_j,\, c_j))$
$\mathit{not\ co\ existence}(t_i,\, t_j)$	$\mathit{not\ responded\ existence}(t_i,\, t_j) \wedge$ $\mathit{not\ responded\ existence}(t_j,\, t_i)$

4.3.2 Soundness in Declare

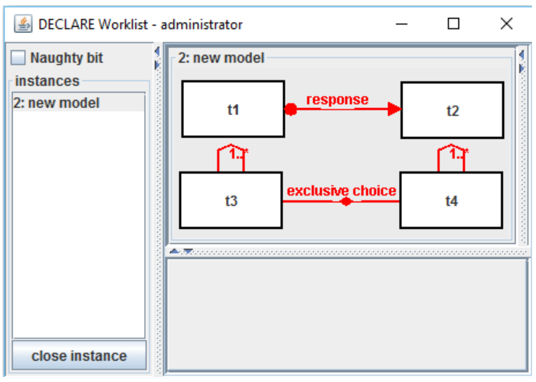
Declare can perform process check to identify whether the process violates option to complete and *no dead task* (MAGGI et al., 2011; MAGGI et al., 2011; MAGGI; MOOIJ; AALST, 2011; PEŠIĆ; BOŠNAČKI; AALST, 2010; PESIC; SCHONENBERG; AALST, 2007; PEŠIĆ; BOŠNAČKI; AALST, 2010). Figure 25(a) presents process *Example1* modeled in Declare. This process has four tasks: t_1, t_2, t_3, t_4; and four constraints: *response*(t_1,t_2), *exclusive choice*(t_1,t_2), *existence*(t_3), *existence*(t_4). The constraint *response*(t_1,t_2) defines that if t_1 is completed then t_2 must be completed afterward, the constraint exclusive *choice*(t_3,t_4) makes completing either exclusively t_3 or t_4 mandatory, but never both of them. However, constraint *existence*(t_3) defines that t_3 must be completed at least once and constraint *existence*(t_4) defines that t_4 must be completed at least once, resulting in a conflict between constraints *exclusive choice*(t_3,t_4), *existence*(t_3) and *existence*(t_4). Given that the conflict existing among constraints means there is no sequence of events that complies with the complete set of constraints, this results in a violation of option to complete. This violation is notified by a There is a conflict message. Figure 25(b) presents process *Example1* at a simulation interface. Since process *Example1* violates *option to complete*, no sequence of events can comply fully with set of constraints, so no sequence of events is enabled and this is notified by displaying the graphics that represent the constraints in red.

Figure 26(a) presents process *Example2* modeled in Declare. This process has four tasks: t_1, t_2, t_3, t_4; and three constraints: *response*(t_1,t_2), *exclusive choice*(t_3,t_4), *existence*(t_3). Constraint *response*(t_1,t_2) defines that if t_1 is completed then t_2 must be completed after that, constraint *exclusive choice*(t_3,t_4) makes completing exclusively either t_3 or t_4, but never both of them, mandatory and constraint *existence*(t_3) defines that t_3 must



Errors
Activity "t4" is dead
There is a conflict
Activity "t3" is dead

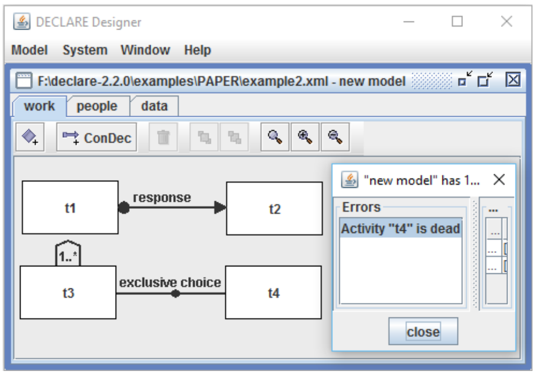
(a) Process *Example1*



(b) Process *Example1* at simulation interface

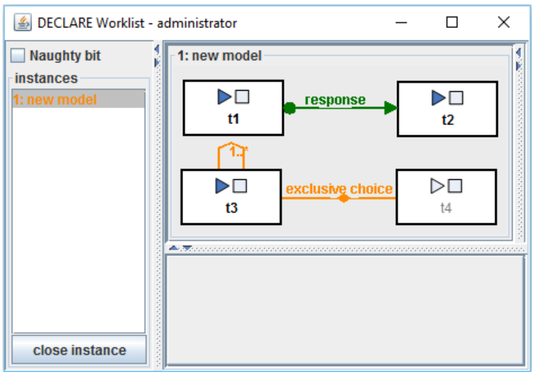
Figura 25 – Process *Example1* at simulation interface

be completed at least once. The combination of constraints *exclusive choice*(t_3,t_4) and *existence*(t_3) make it impossible to complete t_4 . If t_4 is part of the process but cannot be completed, then t_4 is a dead task, so the *no dead task* is violated and this violation is notified by an *Activity* t_4 *is dead* message. Figure 26(b) presents process *Example2* at the simulation interface. As in process *Example2*, t_4 is a dead task that cannot be completed, so this condition is notified, at the simulation interface, by putting in a gray triangle in the squares inside the box that represents a task.



Errors
Activity "t4" is dead

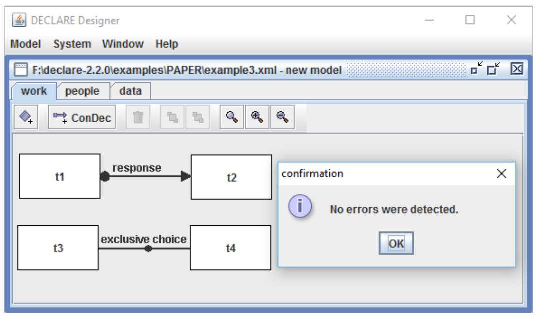
(a) Process *Example2* modeled by Declare



(b) Process *Example2* at simulation interface

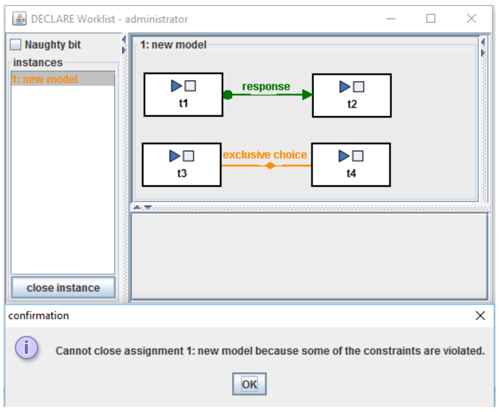
Figura 26 – Process *Example2* modeled by Declare and at simulation interface

Figure 27(a) presents process *Example3* modeled in Declare. This process has four tasks, t_1, t_2, t_3, t_4, and two constraints, *response*(t_1,t_2), *exclusive choice*(t_3,t_4). Constraint *response*(t_1,t_2) defines that, if t_1 is completed, then $t2$ must be completed afterward, constraint *exclusive choice*(t_3,t_4) makes completing either t_3 or t_4 mandatory, but never both of them. The combination of constraints *response*(t_1,t_2) and *exclusive choice*(t_3,t_4) does not violate option to complete and no dead task and this non-violation status is informed by a message of No errors were detected. Figure 27(b) presents a state at the simulation interface of process *Example3*. In the state of process *Example3* displayed in Figure 27(b), t_3 must be completed and thus *exclusive choice*(t_3,t_4) is violated, this means that option to complete is being violated and this violation is notified by inserting an orange graphic representing constraint *exclusive choice*(t_3,t_4). Also, should users try to finish this instance of *Example3*, since option to complete is being violated, this action is not allowed and this is informed by a *Cannot close assignment 1: new model because some the constraints are violated* message. Figure 27(c) presents another state of process *Example3* at the interface simulation. In the state of process *Example3* displayed in Figure 27(c), t_3 was started but has not been not completed or canceled yet, this means that *proper completion* is being violated. At same time that *proper completion* is being violated, the user tries finish the instance of *Example3*, this action of finishing the process is allowed by Declare and the user is advised by a *Are you sure you want to close assignment 1: new model?* message. If the user chooses *YES*, *Example3* is finished.



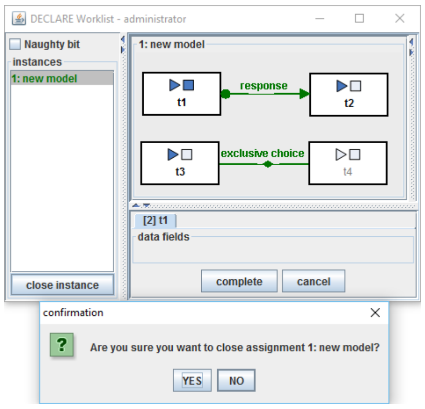
No errors were detected.

(a) Process *Example3* modeled by Declare



Cannot close assignment 1: new model because some of the constraints are violated.

(b) Process *Example3* at simulation interface (*option to complete* is not violated)



Are you sure you want to close assignment 1: new model?

(c) Process *Example3* at simulation interface (*proper completion* can be violated)

Figura 27 – Process *Example3* modeled by Declare and at simulation interface

4.3.3 Constraint violation and enabled events in Declare

At the simulation interface, Declare does not advise the tasks pending in complying with the constraints, instead it informs constraints that have been complied with and those in violation. Declare does this by putting respectively in green and in orange, the graphics that represent them (PESIC; SCHONENBERG; AALST, 2007). Figures 25(b) and 26(b) show this condition. Also, at the interface simulation, Declare informs the enabled and disabled events in a process. To inform the enabled and disabled events Declare uses triangles and squares placed inside the box that represents the task in case in point. The triangle represents the task start event and the square represents the task complete or cancel event. If the graphic is in blue, the respective event has been enabled. If the graphic is in gray, the respective event has been disabled. For example, in Figure 27(c), events *start* t_1, *complete* t_1 *cancel* t_1, *start* t_2 and *start* t_3 are enabled and the events *complete* t_2 *cancel* t_2, *complete* t_3 *cancel* t_3, *start* t_4, *complete* t_4, *cancel* t_4 have been disabled.

4.4 DCR graphs

This subsection presents the main grounds of the DCR graphs framework and how it deals with soundness, enabled events and pending events.

4.4.1 Tasks and constraints in DCR graphs

Dynamic Condition Response (DCR) graphs is a formalism based on discrete event systems applied in modeling constraint-based processes (MUKKAMALA, 2012). The structures defined in DCR allow sets of events and sets of relations among them to be defined. The sequences of events must follow set of relations, i.e. only the sequences that obey the relations are allowed to occur. DCR formalism is based on a collection of three sets: Include (In), Response (Re) and Executed (Ex). These three sets define the process markers. After the execution of an event, these sets may be changed and, consequently, new markers reached. There are three relations that make changes in these sets: include, exclude, and response. Relation *a include b* ($a{\to}{+}b$) defines that if event a is executed, then event b is inserted into the set In. Relation *a exclude b* ($a{\to}\%b$) defines that if event a is executed, then event b is excluded from set In. Relation *a response b* ($a\bullet\to b$) defines that if event a is executed, then event b is inserted into set Re. Events in set Re are pending events. Relation *a condition b* ($a{\to}{\bullet}b$) defines that event a in set Ex is a necessary (but not sufficient) condition to enable event b. Relation *a milestone b* ($a{\to}{\diamond}b$) defines that if event a is in set In, then event b can be executed only after event a is executed. Relation *exclude a* ($\%a$) defines that event a is excluded from set In at the process onset marking. Relation *response a* ($!a$) defines that event a is inserted into set Re at the process onset marking. Table 6 presents the seven DCR relations with their representations, LTL expressions and description.

Event e is enabled at some markers if e is in set In and the intersection of set of events that precedes e ($\to{\bullet}e$) and set In is contained in set Ex, as demonstrated by the following expression:

$$\text{event } e \text{ is enabled} \Leftrightarrow (e \in \text{In}) \wedge ((\text{In} \cap \to{\bullet}e) \subseteq \text{Ex}).$$

Tabela 6 – Five relations from DCR Graphs

Relation	Representation	Description	Expression
a include b	$a{\rightarrow}{+}b$	If the event a is executed then the event b is inserted into the set In	$\text{In} = \text{In} \cup \{b\}$.
a exclude b	$a{\rightarrow}\%b$	If the event a is executed then the event b is excluded from the set In	$\text{In} = \text{In} \setminus \{b\}$.
a condition b	$a{\rightarrow}{\bullet}b$	$a \in \text{Ex}$ is necessary (but not sufficient) condition to the event b to be enabled.	——
a response b	$a{\bullet}{\rightarrow}b$	if the event a is executed then the event b is inserted in the set Re.	$\text{Re} = \text{Re} \cup \{b\}$.
a milestone b	$a{\rightarrow}{\diamond}b$	if the event a is in the set $\text{In} \cap \text{Re}$ then the event b can be executed only after the event a is executed.	——
exclude a	$\%a$	The event a does not pertain to the set In at the initial marking	$a \notin \text{In}$, at the initial marking.
response a	$!a$	The event a pertains to the set Re at the initial marking	$a \in \text{Re}$, at the initial marking.

Event e is pending (mandatory execution) at some marker, if e is in the intersection of sets In and Re, as shown by the following expression:

$$\text{event } e \text{ is pendent} \Leftrightarrow e \in \text{Re} \cap \text{In}.$$

The execution of an event e defines that event e is inserted in set Ex and event e is excluded from set Re:

$$\text{event } e \text{ is the last one that was executed} \Leftrightarrow (\text{Ex} = \text{Ex} \cup \{e\}) \wedge (\text{Re} = \text{Re} \setminus \{e\}).$$

Every relation or logical condition for every event e is valid only if event e is in set In ($e \in \text{In}$), if event e is not in set In ($e \notin \text{In}$) then all of its relations or logical conditions are annulled.

4.4.2 Soundness in DCR graphs

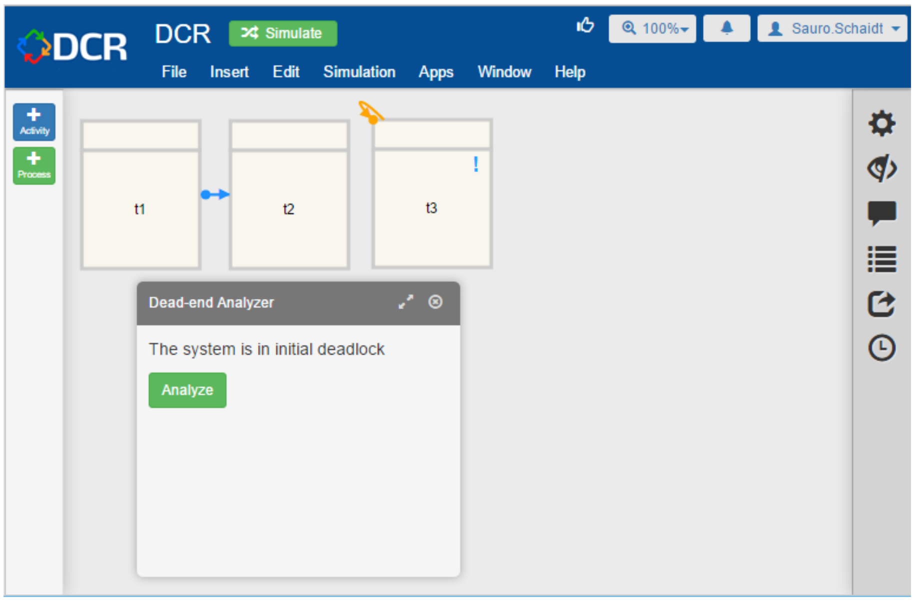
DCR frameworks can perform process checks in order to identify whether the process violates *option to complete* and *no dead task*.

Figure 28(a) presents process *Example4* modeled in DCR. This process has three tasks: t_1, t_2, t_3; and three constraints: *response*(t_1,t_2) (t_1●→t_2), *condition*(t_3,t_3) (t_3→●t_3), *response*(t_3) (!t_3). Constraint *response*(t_1,t_2) defines that, if t_1 is completed, then t_2 must be completed after that, constraint *condition*(t_3,t_3) defines that $t3$ must be executed before t_3, constraint *response*(t_3) defines that t_3 pertains to set Re at the initial marking. The consequence of constraint *condition*(t_3,t_3) is that t_3 will never be executed. But t_3 pertains to set In at the initial marking and constraint *response*(t_3) defines that $t3$ pertains to set Re at the initial marking, then t_3 pertains to the intersection of Re and In ($t_3 \in \text{Re} \cap \text{In}$) and thus execution of t_3 is mandatory. So the general condition is such that t_3 will never be executed because of constraint *condition*(t_3,t_3), and at same time, execution of t_3 is mandatory because of constraint *response*(t_3). The result is a conflict. There is no sequence of events capable of complying with this set of constraints and, therefore, there is an option to complete violation. This is notified by message System is in initial deadlock . Figure 28(b) presents process *Example4* at the simulation interface which also allows checking process Example4 and obtaining the System is in initial deadlock message again (MUKKAMALA, 2012).

Figure 29(a) shows process *Example5* modeled by DCR. This process has three tasks: t_1, t_2, t_3; and two constraints: *response*(t_1,t_2), *condition*(t_3,t_3). Constraint *response*(t_1,t_2) defines that, if t_1 is completed, then t_2 must be completed after that, constraint *condition*(t_3,t_3) defines that t_3 must be executed before t_3. The consequence of constraint *condition*(t_3,t_3) is that t_3 will never be executed. Since t_3 cannot be executed, t_3 is a dead task and this results in a *no dead task* violation. This is displayed by the graphical user interface, by putting using gray lines around the box that represents the task. Since there is no violation of *option to complete*, the framework displays a *This graph cannot reach a dead-end* message. Figure 29(b) presents process *Example5* at the simulation interface. The simulation interface also enables checking process *Example5* and obtaining a *This graph cannot reach a dead-end* message again (MUKKAMALA, 2012).

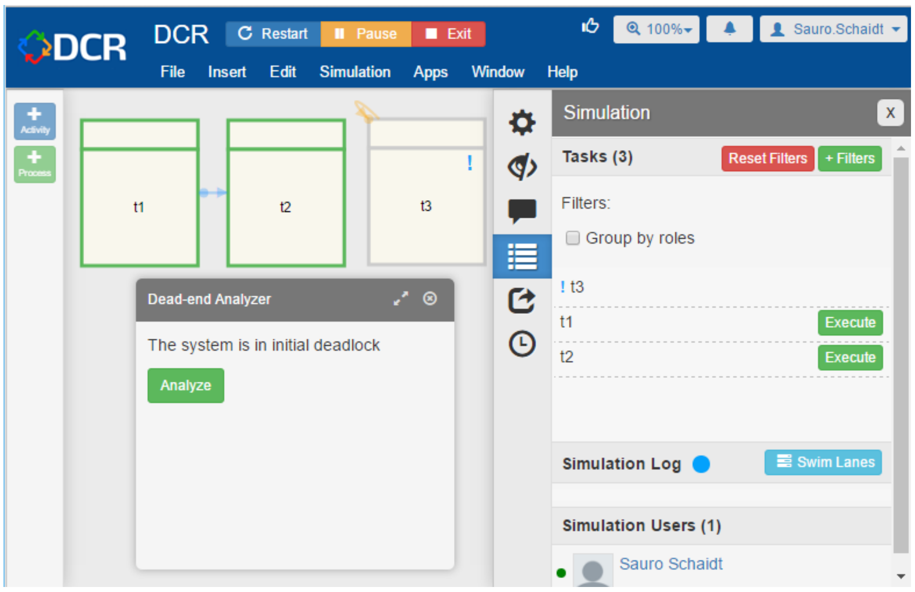
4.4.3 Enabled and pendent events in DCR graphs

In order to inform any existing pending and enabled events, DCR uses the list of events that is at the right side of the simulation window. If the task is enabled then the green button *Execute* is displayed next to the name of the task. If the task is not enabled, no button is displayed. If the task is pending, the symbol "!"is displayed next to the name of the task. If the task is not pending, no symbol is shown (MUKKAMALA, 2012). For example, in Figure 28(b), tasks t_1, t_2 are enabled but are not pending, whereas task t_3 is



The system is in initial deadlock

(a) Process *Example4* modeled by DCR



The system is in initial deadlock

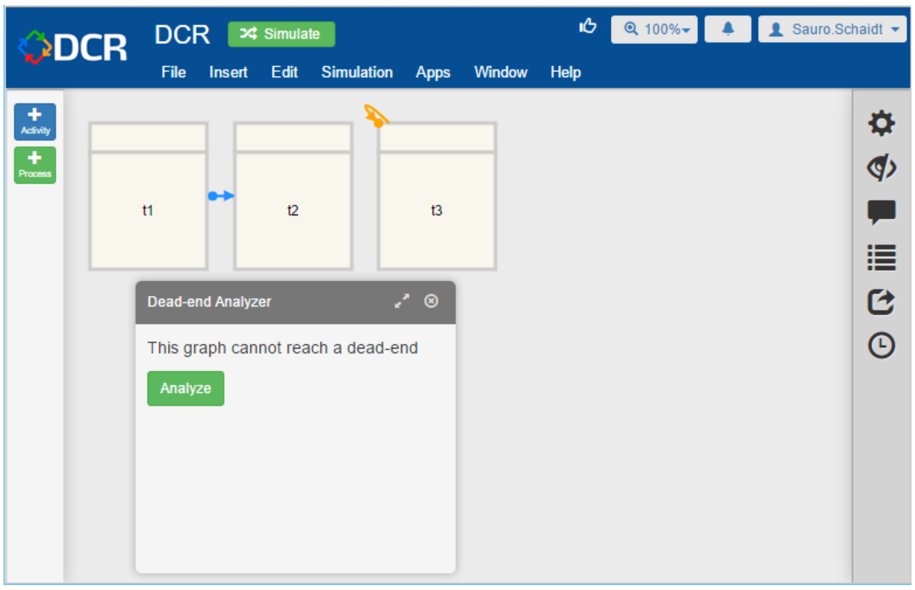
(b) Process *Example4* at simulation interface

Figura 28 – Process *Example4* modeled by DCR and at simulation interface

not enabled, but is pendent.

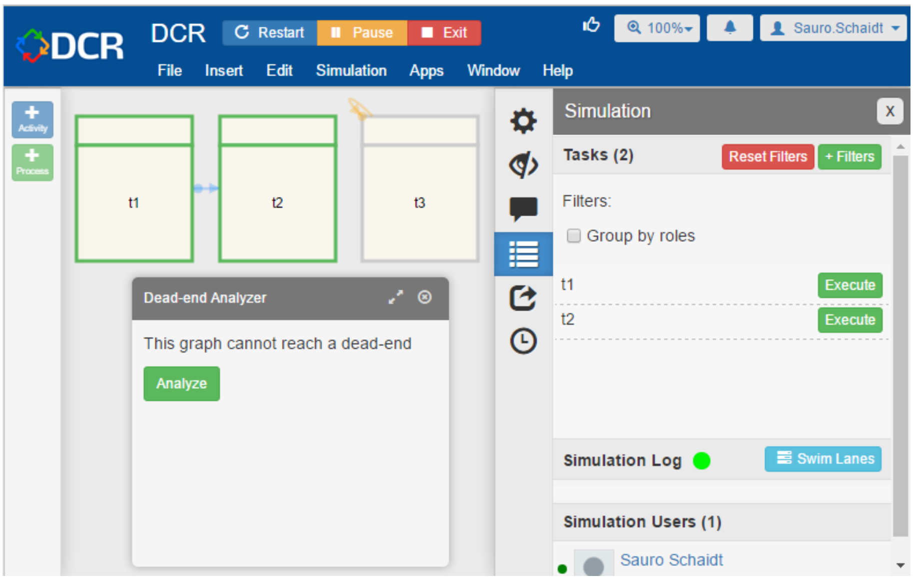
4.5 SCT approach

The Supervisory Control Theory (SCT) is mathematical formalism for automatic synthesis of optimal controllers for discrete event systems (DES) (RAMADGE; WONHAM, 1987; RAMADGE; WONHAM, 1989; WONHAM; RAMADGE, 1987). SCT presumes that a set of tasks may have an uncontrollable behavior that might violate some of the properties required. This behavior must be modified through an agent, the supervisor, in order to



This graph cannot reach a dead-end

(a) Process *Example5* modeled by DCR



This graph cannot reach a dead-end

(b) Process *Example5* at simulation interface

Figura 29 – Process *Example5* modeled by DCR and at simulation interface

achieve a set of specifications or to ensure that certain restrictions are not violated. The supervisor acts on the set of tasks specified by preventing the generation of some events and allowing others. The supervisors are obtained through a set of logical operations. The techniques used in SCT to obtain supervisors also may be used for modeling constraint based processes in such a way the supervisor is equivalent to the process modeled (SANTOS et al., 2014).

4.5.1 Tasks and constraints in SCT approach

SCT approach provides a set of automata to represent task behaviors in a process and the constraints that must be imposed on them (SANTOS et al., 2011; SANTOS et al., 2014; SCHAIDT et al., 2013; CESTARI et al., 2014; SCHAIDT et al., 2013). Figure 30 presents a task automaton in the SCT approach. The automaton shown in Figure 30 has three events: start the task (s), complete the task (c) and cancel the task (x), if the task is started (s is executed), then either completing the task (executing c) or canceling it (executing x) is mandatory, but not both of them. With respect to controllability of events, the start event is controllable, but the *complete* and *cancel* events are not. Controllable events are those that users have the power to decide whether they must be executed or not. Uncontrollable events are those that users do not have the power to decide whether they must be executed or not. Figure 31 presents some of constraints provided in the SCT approach. The behavior of each constraint in Figure 31 has already been described in Table 4.

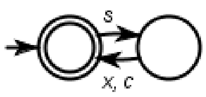
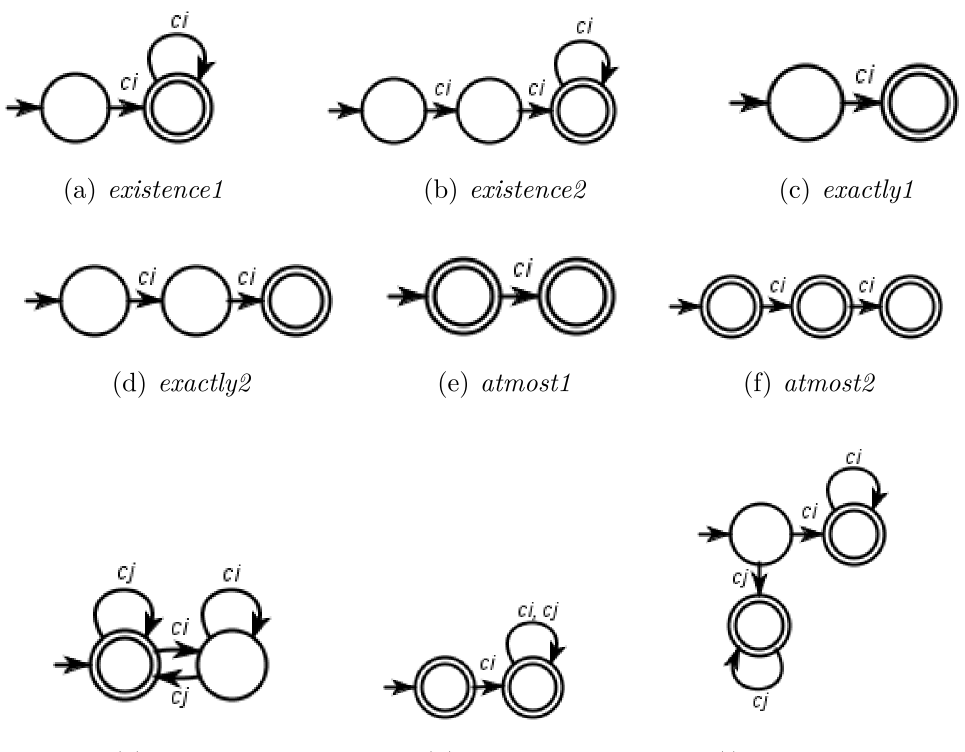


Figura 30 – Task model proposed by (SANTOS et al., 2014): event *start* (s) is controllable, and events *complete* (c) and *cancel* (x) are uncontrollable



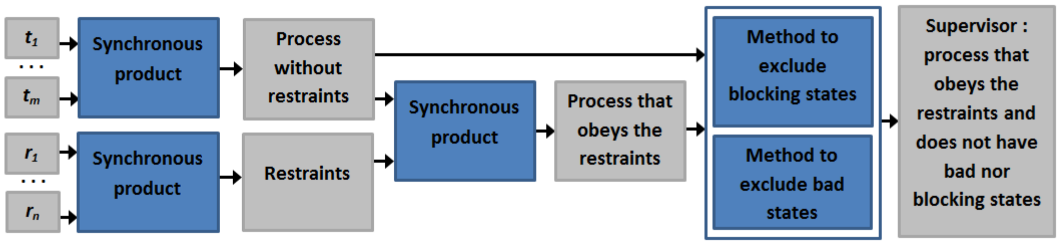
(a) *existence1* (b) *existence2* (c) *exactly1*

(d) *exactly2* (e) *atmost1* (f) *atmost2*

(g) *response* (h) *precedence* (i) *exclusive1of2*

Figura 31 – Constraints SCT

Figure 32 shows the sequence of operations to synthesize the supervisor automaton. Initially there is a set with m tasks ($t_1 ... t_m$) and another with n constraints ($r_1 ... r_n$). The synchronous product of the set of tasks generates *Process without constraints*, and the synchronous product of the set of constraints generates constraints. The synchronous product of *Process without constraints* and constraints generates *Process with constraints*. The method to exclude blocking states and the method to exclude bad states make successive comparisons between *Process without constraints* and *Process with constraints* to generate the supervisor. The supervisor automaton is equivalent to the automaton of the process that obeys constraints and does not have blocking nor bad states.



t_1
t_m
r_1
r_n
Synchronous product
Synchronous product
Process without restraints
Restraints
Synchronous product
Process that obeys the restraints
Method to exclude blocking states
Method to exclude bad states
Supervisor : process that obeys the restraints and does not have bad nor blocking states

Figura 32 – Sequence of operations for reaching the supervisor

The formal definition of synchronous product (parallel composition) will be presented next. Consider the two automata

$$G_1 = \{Q_1, \Sigma_1, \delta_1, \Gamma_1, q_{01}, Q_{m1}\}$$

$$G_2 = \{Q_2, \Sigma_2, \delta_2, \Gamma_2, q_{02}, Q_{m2}\}$$

The parallel composition (or synchronous product) of G_1 and G_2 is the automaton

$$G_1 || G_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{12}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2}), \text{ where}$$

$$\delta_{12}\left(\left(q_1, q_2\right), e\right) := \begin{cases} \left(\delta_1\left(q_1, e\right), \delta_2\left(q_2, e\right)\right) & if\ e \in \Gamma_1\left(q_1\right) \cap \Gamma_2\left(q_2\right) \\ \left(\delta_1\left(q_1, e\right), q_2\right) & if\ e \in \Gamma_1\left(q_1\right) \backslash \Sigma_2 \\ \left(q_1, \delta_2\left(q_2, e\right)\right) & if\ e \in \Gamma_2\left(q_2\right) \backslash \Sigma_1 \\ undefined, & otherwise \end{cases} \tag{4.1}$$

In parallel composition, a common event, that is, an event in $\Sigma_1 \cap \Sigma_2$, can only be executed if both automata execute it simultaneously. Thus, the two automata are "synchronized"for events in common. Private events, that is, those in $(\Sigma_2 \backslash \Sigma_1) \cup (\Sigma_1 \backslash \Sigma_2)$, are not subject to this constraint and can be executed whenever possible. In this kind of interconnection, a component can execute its private events without requiring 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 must 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 G_1 and G_2 shuffle.

The method to exclude blocking states identifies the states for which there is no sequence of events that allows reaching a marker state and excludes these states and the events that lead to them. The method to exclude bad states identifies the states where uncontrollable events are being disabled and excludes these states and the events that lead to them. The methods to exclude blocking states and to exclude bad states works in alternation until there be are no blocking states and no bad states left (RAMADGE; WONHAM, 1987; WONHAM; RAMADGE, 1987; RAMADGE; WONHAM, 1989).

4.5.2 Soundness, Enabled and pendent events in SCT approach

Since the SCT approach is not a framework, just a mathematical formalism, the paper will not analyze soundness properties and enabled and pendent events in the SCT approach.

4.6 Simple Declarative Language (SDL)

This section introduces the Simple Declarative Language (SDL), a conceptual framework for modeling constraint based processes. The SDL framework provides a single task model that is used in every process modeled. This task model is the same as in the SCT approach. Figure 30 shows the task automaton in SDL. The behavior of the task automaton model is described in Section 5. The SDL framework provides for controllable task start, complete and cancel events. SDL does not provide uncontrollable events. The SDL framework provides a set of three constraints: existence, precedence, and response. These constraints models are the same as in the SCT approach. Figures 31(a), 31(g) and 31(h), respectively, show the automata for *atleast1*(t_i), *response*(t_i,t_j), *precedence*(t_i,t_j) used in SDL. The behavior of constraints automata models are described in Table 4. The next items introduce the features of an SDL framework in design and run times.

4.6.1 Design time

At design time, the SDL framework checks compliance of sets of tasks and constraints with the syntactic and semantics rules defined in the SDL language. To this end, the SDL framework provides the *Method to check syntax and semantics of SDL language*. Figure 33 shows the block diagram with the *Method to check syntax and semantics of SDL language*.

The syntax rules in SDL are described in section 4.6.3. *Method to check syntax and semantics of SDL language* checks whether given *Set of tasks and constraints* fulfills the conditions described in section 4.6.3. If those conditions are met by the *Set of tasks* *and constraints* then the process is syntactically correct, otherwise the process is not syntactically correct.

The semantics rules in SDL are described in section 4.6.4. The *method to check syntax and semantics of SDL language* checks whether a given *Set of tasks and constraints* fulfills the conditions described in section 4.6.4. If those conditions are met by the set of tasks and constraints, the process is semantically correct, otherwise the process is not semantically correct.

If the syntactic and semantic rules are followed correctly, the *Set of tasks and constraints* is valid. Otherwise the *Set of tasks and constraints* is not valid. In other words, in SDL frameworks, a process is valid if, and only if, the *Set of tasks and constraints* comprising the process is syntactically and semantically correct.

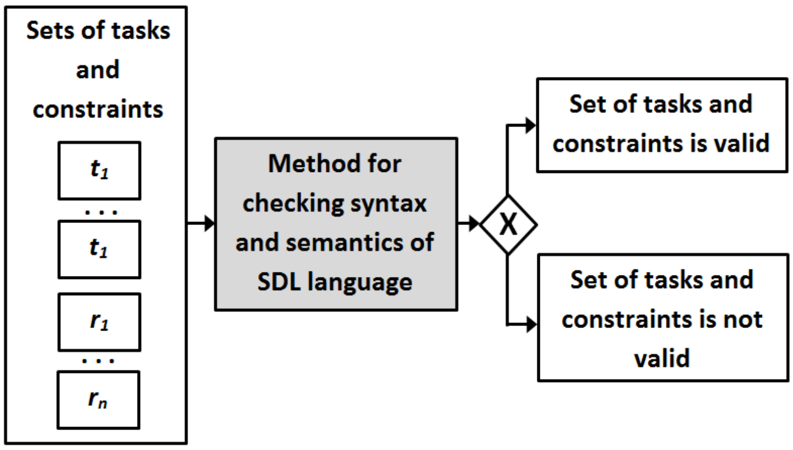

Sets of tasks and constraints
t1
...
t1
r1
...
rn
Method for checking syntax and semantics of SDL language
X
Set of tasks and constraints is valid
Set of tasks and constraints is not valid

Figura 33 – The single method of SDL framework at design time

4.6.2 Run time

Next, the sequence of operation performed by an SDL framework at run time is explained. Figure 34 shows the block diagram of the operations at run time in an SDL framework.

At run time, the *Valid set of tasks and constraints* is the input for the *Method to calculate pendent events* and *Method synchronous product*. The *Method to calculate pending events* is set out in section 4.6.6. The *Method synchronous product* is presented in section 4.5.1. The *Method to calculate pending events* applies the valid *Set of tasks and constraints* to calculate the process's set of pending events. The *Method synchronous product* applies the *Set of tasks and constraints* to calculate the process's set of enabled events. From *Set of enabled events*, users choose whether to execute a given event. When users choose to execute an event, *Method to update sequence of events executed* updates its outputs: *Sequence of events executed* and *Last event executed.*

Method to update the current state in the automata takes the *Last event executed* to update the current state of each automata in the *Set of tasks and constraints*. After the current state of each automaton in the *Set of tasks and constraints* has been updated, a new iteration is started, i.e. *Method to calculate pendent events* takes set of tasks and constraints to calculate *Set of pending events* of the process. *Method synchronous product* applies the *Set of tasks and constraints* to calculate *Set of enabled events* of the process. This procedure continues until the process is finished.

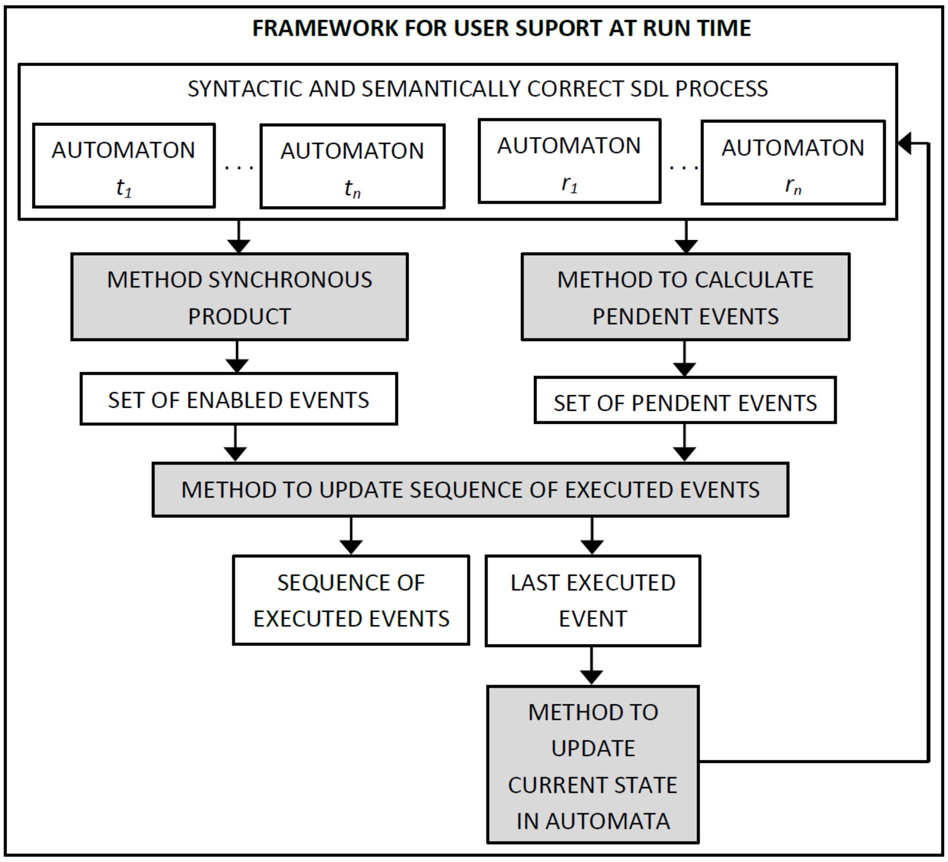

FRAMEWORK FOR USER SUPORT AT RUN TIME
SYNTACTIC AND SEMANTICALLY CORRECT SDL PROCESS
AUTOMATON t_1
AUTOMATON t_n
AUTOMATON r_1
AUTOMATON r_n
METHOD SYNCHRONOUS PRODUCT
METHOD TO CALCULATE PENDENT EVENTS
SET OF ENABLED EVENTS
SET OF PENDENT EVENTS
METHOD TO UPDATE SEQUENCE OF EXECUTED EVENTS
SEQUENCE OF EXECUTED EVENTS
LAST EXECUTED EVENT
METHOD TO UPDATE CURRENT STATE IN AUTOMATA

Figura 34 – The three methods of SDL framework at run time

4.6.3 Syntax in SDL

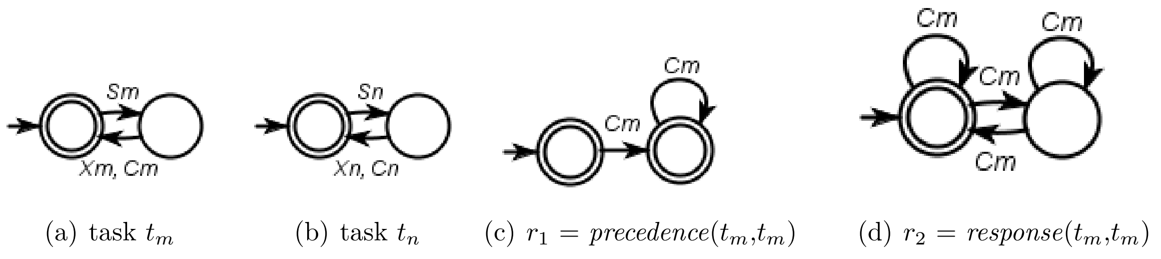
A process modeled by SDL is syntactically correct when it complies with Definition 4.6.1 that is presented next.

Definition 4.6.1. *A process modeled by SDL is a pair $P = (T, R)$, such that*

- *$T = \{t_1,...,t_n\}$ is a finite set of tasks.*
- *$R = \{r \mid r = atleast1(t_i)$ or $r = response(t_i,t_j)$ or $r = precedence(t_i,t_j)\}$, or $R = \emptyset$, is a finite set of constraints.*
- *If $atleast1(t_i) \in R$ then $t_i \in T$.*
- *If $response(t_i,t_j) \in R$ then $t_i \in T$, $t_j \in T$, $t_i \neq t_j$.*

- *If precedence(t_i,t_j) $\in$ R then $t_i \in T$, $t_j \in T$, $t_i \neq t_j$.*

The behavior of constraints *response*(t_i,t_j) and *precedence*(t_i,t_j) when $t_i = t_j$ is presented next. To do this, the tasks and constraints shown in Figure 35 will be used.



(a) task t_m (b) task t_n (c) r_1 = *precedence*(t_m,t_m) (d) r_2 = *response*(t_m,t_m)

Figura 35 – Some tasks and constraints

Considering task t_m and constraint $r1$ = *precedence*(t_m,t_m), Figures 35(a) and 35(c), such that $t_m \in$ T and $r_1 \in$ R. The synchronous product of t_m and r_1 is automaton $t_m||r_1$ in Figure 36. The minimization of $tm||r1$ is the automaton tm in Figure 35(a). So $t_m||r_1$ does not impose any change in process behavior, merely increasing the number of states in the process, which is not desirable. Thus, *precedence*(t_m,t_m) is not a valid constraint in SDL framework.

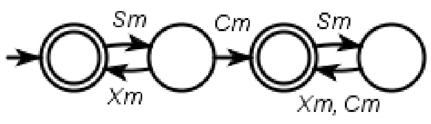


Figura 36 – $t_m||r_1$

Considering task t_m and constraint r_2 = *response*(t_m,t_m), Figures 35(a) and 35(d), such that $t_m \in$ T and $r_2 \in$ R. The synchronous product of t_m and r_2 is automaton $t_m||r_2$ in Figure 37. The minimization of $t_m||r_2$ is automaton t_m in Figure 35(a). So $t_m||r_2$ does not impose any change in process behavior, merely increasing the number of states in the process, which is not desirable. Thus, *response*(t_m,t_m) is not a valid constraint in SDL framework.

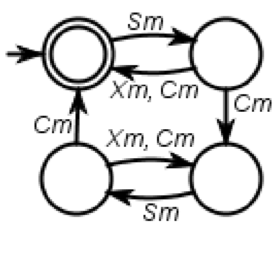


Figura 37 – $t_m||r_2$

Example 4.6.1 with 3 processes that violate the syntax rules of the Definition 6.1. is shown next.

Example 4.6.1. *The process with T = {t_1, t_2, t_3, t_4} and R = atleast1(t_1), response(t_2,t_3), exclude(t_3,t_4)} is syntactically incorrect because exclude(t_3,t_4) is not a valid constraint in SDL. The process with T = {t_1, t_2, t_3, t_4} and R = {atleast1(t_5), response(t_2,t_3), precedence(t_3,t_4)} is syntactically incorrect because t_5 is not a task in T. The process with T = {t_1, t_2, t_3, t_4} and R = {atleast1(t_1), response(t_2,t_3), precedence(t_4,t_4)} is syntactically incorrect because $t_i = t_j$ in precedence(t_4,t_4)}.*

4.6.4 Semantics in SDL

Semantics in SDL is related to soundness. An SDL process is semantically correct if, and only if, it complies with soundness, which is Definition 4.6.2 presented next.

Definition 4.6.2. *A SDL process is semantically correct if, and only if, it complies with soundness.*

So, it becomes necessary to define the concept of soundness in SDL processes. Definition 4.2.3 establishes the concept of soundness for workflow nets. In Definition 4.2.3, the requirements *option to complete*, *no dead task* and *proper completion* are described in terms of transitions and places since they are the basic constructs in workflow nets. Although the fundamentals of *soundness* are the same for any language, the requirements must be described in terms of the language that is being used to model the process. Therefore Definition 4.2.3 must be drafted in terms of SDL constructs. SDL frameworks have four basic constructs: *task*, *atleast1*, *precedence* and *response*. Each of these constructs is represented by an automaton. This implies in the need to rewrite Definition 4.2.3 in terms of the SDL automata in order to define the *soundness* concept as applicable to the SDL language. Definition 4.6.3 does this and is set out next.

Definition 4.6.3. *A SDL process complies with soundness if and only if it complies with three requirements:*

- *option to complete: From any state of the process is possible to reach a marked state.*
- *no dead task: For each task in the process, there is a sequence of events in which the task can be completed.*
- *proper completion: If a marked state is reached then every task in the process is not started or is completed or canceled.*

After defining soundness requirements for the SDL language, it becomes necessary to investigate which SDL automata influences which soundness requirements. This is demonstrated in the following subsection.

4.6.4.1 SDL automata and soundness requirements

SDL frameworks apply the method synchronous product to generate the sequences of events in run time. In SDL frameworks, the inputs of the method synchronous product are the automata *task*, *atleast1*, precedence and *response*. Therefore, it becomes necessary to understand how these automata influence *option to complete*, *no dead task* and *proper completion* in the sequences of events generated by the method synchronous product. This is described in the following section.

The method synchronous product was introduced in section 4.5. The method synchronous product receives a set of automata as input and produces sequences of events as output. For each combination of states of the automata in the input, the synchronous product defines a single state in the output. The method synchronous product defines if the state in the output is marked or not marked and defines which set of enabled events applies to this state.

In synchronous product, for each combination of states of the automata in the input, when all the states in the combination are marked, the state in the output is also marked. Otherwise, the state in the output is not marked. There are some unmarked states in the output if, and only if, there are some unmarked states in the method inputs.

Thus, in SDL frameworks, the sequence of events in the output of the method synchronous product can violate *option to complete* if, and only if, there is an automaton in the input that provides an unmarked state. This is the case of *task*, *atleast1* and *response*.

In synchronous product, for each combination of states of the automata in the input, if an event is enabled in all the automata in which it is present in set of events, then this event is enabled in the state of the output. Otherwise, this event is not enabled in the state of the output. There may be disabled events in a state of the output if, and only if, this same event is disabled in some automaton in the input.

Thus, in SDL frameworks, the sequence of events in the output of the method synchronous product can violate no dead task if, and only if, there is an automaton in the input that provides a state in which an event *complete* is disabled. This is the case of *task* and *precedence*.

In SDL frameworks, the sequence of events in the output of the method synchronous product can violate *proper completion* if, and only if, there is an automaton providing the event *start* in the input of the method synchronous product. This is the case of *task*.

Table 7 presents the soundness requirements that can be affected by each of the SDL automata.

Tabela 7 – SDL automata and the soundness requirements that they affect

automaton	option to complete	no dead task	proper completion
task	X	X	X
atleast1	X		
response	X		
precedence		X	

After identifying the SDL automata that influence the soundness requirements, the way synchronous products of SDL automata influence soundness requirements must be investigated. This is demonstrated in the following subsection.

4.6.4.2 Synchronous product of SDL automata and its influence on soundness requirements

This subsection investigated the behavior of synchronous products of the automata on the soundness requirements they affect. This investigation will be performed in two parts. First, the synchronous product of the automata of the task is investigated separately. Second, constraints automata are investigated. The investigation is carried out in this way because the synchronous product of tasks represent the behavior of the process without constraints, and the synchronous products of constraints represent the behavior of process constraints. Thus, tasks and constraints can be investigated separately.

Next, the synchronous product of tasks with respect to *option to complete*, *no dead task* and *proper completion* is investigated.

4.6.4.2.1 Influence of synchronous product of *tasks* on *option to complete*, *no dead task* and *proper completion*

Each automaton in task complies with one of the following: *option to complete*, *no dead task* and *proper completion*. This happens because in *task* automata: 1) a marked state can be reached from any other state, 2) there is a sequence of events in which the *task* can be completed, 3) if the marked state is reached then either the *task* is not started, or it is completed or canceled.

The synchronous product of a set of tasks produces sequences of events that also comply with *option to complete*, *no dead task* and *proper completion*. This happens because the automata of tasks do not have common events among them. Therefore, the synchronous product of a set of tasks does not change the behavior of each task separately. The result is that each *task* automaton complies with *option to complete*, *no dead task* and *proper completion* and, consequently, the synchronous product of all the tasks also complies with *option to complete*, *no dead task* and *proper completion*.

The synchronous product of a set of tasks guarantees that *proper completion* is achieved for any set of tasks but also for any set of constraints. This happens because the *task* automaton establishes that an unmarked state is reached whenever a *start* event is executed. So, the synchronous product of a set of tasks and any set of constraints always produces sequences of events in which, whenever a *start* event is executed, an unmarked state is reached.

After investigating the *soundness* requirements that can be affected by the synchronous product of the SDL tasks, the synchronous product of the SDL constraints must be investigated with respect to the *soundness* requirements they can affect. This is done in the following subsections.

4.6.4.2.2 Influence of synchronous product of constraints *precedence* on *no dead task*

Every automaton of constraint *precedence* complies with *no dead task*. This happens because for any *task* related to a *precedence* automaton, there is a sequence of events in which the *task* can be completed.

It is possible for a *complete* event to be common to two different *precedence* constraints. Therefore, a set of *precedence* constraints can drive two kinds of combination: *sequence of precedences* and *loop of precedences*. Next, Definitions 4.6.4 and 4.6.5 present these combinations.

Definition 4.6.4. *Let $P = \{T, R\}$ be a SDL process. A sequence of constraints precedence is a set $S_p \subseteq R$, such that*

- *if $|S_p|$ = 1 then S_p = {precedence(t_i,t_j)}.*
- *if $|S_p|$ = 2 then S_p = {precedence(t_i,t_j), precedence(t_j,t_k)}.*
- *if $|S_p|$ = 3 then S_p = {precedence(t_i,t_j), precedence(t_j,t_k), precedence(t_k,t_l)}.*
- *etc...*

Definition 4.6.5. *Let $P = \{T, R\}$ be a SDL process. A loop of constraints precedence is a set $L_p \subseteq R$, such that*

- *if $|L_p|$ = 2 then L_p = {precedence(t_i,t_j), precedence(t_j,t_i)}.*
- *if $|L_p|$ = 3 then L_p = {precedence(t_i,t_j), precedence(t_j,t_k), precedence(t_k,t_i)}.*
- *etc...*

The synchronous product from automata in sequences of constraints precedence produces sequences of events in which each task can be completed at least once. For example, if $|S_p| = 3$ then $S_p =$ *precedence*(t_i,t_j), *precedence*(t_j,t_k), *precedence*(t_k,t_l). The synchronous product of Sp establishes that the sequence to complete the tasks is t_i, t_j, t_k, t_l. After t_i is completed for the first time, t_i can be completed infinite times and t_j can be completed for the first time. After t_j is completed for the first time, t_i, t_j can be completed infinite times and t_k can be completed for the first time. After t_k is completed for the first time, t_i, t_j, t_k can be completed infinite times and t_l can be completed for the first time. After t_l is completed for the first time, t_i, t_j, t_k, t_l can be completed infinite times. This reasoning is valid for any S_p. So, the synchronous product from *sequences of precedence constraints* comply with *no dead task*. Thus, *sequences of precedence constraints* are allowed to exist in the process.

The synchronous product from automata in loops of *precedence* constraints produces sequences of events in which the tasks cannot be completed. For example, if $|L_p| = 3$ then L_p = *precedence*(t_i,t_j), *precedence*(t_j,t_k), *precedence*(t_k,t_i) can be presumed. The synchronous product of L_p defines that t_i must precede t_j, t_j must precede t_k, t_k must precede t_i. In other words, no task in L_p can be completed because the execution of the *complete* event for every task must be preceded by the execution of a *complete* event in some other task. There is no "free" *complete* event to be executed before all the others. This reasoning is valid for any L_p. This means the synchronous product from *loops of precedence constraints* violate *no dead task*. Thus, *loops of precedence constraints* must be avoided.

4.6.4.2.3 Influence of synchronous product of constraints *response* on *option to complete*

Each automaton of constraint response complies with option to complete. This happens because from any state of the response automaton, it is possible to reach a marked state.

A complete event can be common to two different response constraints. So a set of response constraints can make two kinds of combination: sequence of responses and loop of responses. In the following, Definitions 4.6.6 and 4.6.7 present these combinations.

Definition 4.6.6. *Let $P = \{T, R\}$ be a SDL process. A sequence of constraints response is a set $S_r \subseteq R$, such that*

- *if $|S_r| = 1$ then S_r = {response(t_i,t_j)}.*
- *if $|S_r| = 2$ then S_r = {response(t_i,t_j), response(t_j,t_k)}.*
- *if $|S_r| = 3$ then S_r = {response(t_i,t_j), response(t_j,t_k), response(t_k,t_l)}.*
- *etc...*

Definition 4.6.7. *Let $P = \{T, R\}$ be a SDL process. A loop of constraints response is a set $L_r \subseteq R$, such that*

- *if $|L_r| = 2$ then L_r = {response(t_i,t_j), response(t_j,t_i)}.*
- *if $|L_r| = 3$ then L_r = {response(t_i,t_j), response(t_j,t_k), response(t_k,t_i)}.*

- *etc...*

The synchronous product from automata of sequences of response constraints produces sequences of events wherein a marked state can always be reached from any state of the automaton. For example, supposing $|S_r| = 3$ then $S_r =$ *response*(t_i,t_j), *response*(t_j,t_k), *response*(t_k,t_l). If no task in L_r is completed, then a marked state is reached. If t_i is completed, then the marked state is reached from any state of the synchronous product automaton after t_j,t_k,t_l are completed. If t_j is completed, then the marked state is reached from any state of the synchronous product automaton after t_k , t_l are completed. If t_k is completed then the marked state is reached from any state of the synchronous product automaton after t_l is completed. This reasoning is valid for any S_r. So, the synchronous product from *sequences of response constraints* comply with *option to complete*. Thus, *sequences of response constraints* are allowed to exist in the process.

The synchronous product from automata of loops of response constraints produces sequences of events wherein it is impossible to reach a marked state from any state of the automaton. For example, one supposes $|L_r| = 3$ then $L_r =$ *response*(t_i,t_j), *response*(t_j,t_k), *response*(t_k,t_i). If t_i is completed then t_j must be completed. If t_j is completed then t_k must be completed. If t_k is completed then t_i must be completed and then returns to first execution of complete events, i.e all the tasks must be completed again. This behavior is endless, making it impossible to reach a marked sate in the process because there is always a task to be completed. So, the synchronous product from *loop of response constraints* violates *option to complete*. Thus, *loops of response constraints* must be avoided.

4.6.4.2.4 Influence of synchronous product of constraints *atleast1* on *option to complete*

Every *atleast1* automaton complies with option to complete. This happens because in the *task* automaton a marked state can be reached from any state.

The synchronous product of a set of *atleast1* automata produces sequences of events that are also compliant with option to complete. This happens because *atleast1* automata do not have events in common among them. So, the synchronous product of a set of *atleast1* automata does not change the individual behavior of each *atleast1* automaton. The result is that each *atleast1* automaton remains compliant with option to complete and, consequently, the synchronous product of any set of atleast1 automata is also compliant with option to complete.

4.6.4.2.5 Influence of synchronous product of constraints *response* and *atleast1* on *option to complete*

The synchronous product of a *sequence of response constraints* complies with *option to complete*. The synchronous product from a set of *atleast1* automata complies with *option* *to complete.* The synchronous product of a *sequence of response constraints* and a set of *atleast1* is also compliant with *option to complete.* For example, supposing $|S_r| = 3$ then S_r = *response*(t_i,t_j), *response*(t_j,t_k), *response*(t_k,t_l). If *atleast1*(t_i) is in set *atleast1*, then the marked state is reached from any state of the synchronous product automaton after t_i, t_j, t_k, t_l are completed in this order. If *atleast1*(t_j) is in set *atleast1*, then the marked state is reached from any state of the synchronous product automaton after t_j, t_k, t_l are completed in this order. If *atleast1*(t_k) is in set *atleast1*, then the marked state is reached from any state of the synchronous product automaton after t_k, t_l are completed in this order. If *atleast1*(t_l) is in set *atleast1*, then the marked state is reached from any state of the synchronous product automaton after t_l is completed. This reasoning is valid for any S_r and any set of *atleast1*. So, the synchronous product from a *sequence of response constraints* and a set of *atleast1* complies with *option to complete.* Thus, *sequence of response constraints* and set of *atleast1* are allowed to exist together in the process.

4.6.4.3 Conditions to SDL processes comply with soundness requirements

The previous subsections demonstrated that there are six types of sets that can serve as input for the synchronous product in SDL frameworks: set of *tasks*, set of *atleast1*, *sequences of precedence constraints*, *loops of precedence constraints*, *sequences of response constraints* and *loops of response constraints.*

Theorem 4.6.1. *Every syntactically correct SDL process complies with proper completion.*

Proof: In SDL frameworks, if a set of process tasks and constraints is valid then the process is syntactically correct. This is Definition 4.6.1. In SDL frameworks, a syntactically correct process is comprised of a set of tasks and a set of constraints. This is also Definition 4.6.1. The synchronous product from the set of tasks guarantees that proper completion is always fulfilled for any set of tasks and constraints. This is demonstrated in section 4.6.4.2.1. So, it is possible to establish that if an SDL process is syntactically correct, then it is compliant with proper completion.

Theorem 4.6.2. *Let* $P = \{T, R\}$ *be a syntactically correct SDL process. Let* $L_p \subseteq R$, $L_r \subseteq R$ *be, respectively, any loop of constraints precedence and any loop of constraints response. P complies with option to complete and no dead tasks iff*

- *option to complete:* $L_r = \emptyset$
- *no dead tasks:* $L_p = \emptyset$

Proof: Sections 4.6.4.2.1 and 4.6.4.2.2 demonstrated that no dead task is violated solely by loops of precedence constraints. Therefore, it is possible to establish that SDL processes are not compliant with no dead task if, and only if, they do not have any precedence constraints loops. Sections 4.6.4.2.1, 4.6.4.2.3, 4.6.4.2.4 and 4.6.4.2.5 demonstrated that option to complete is violated only by loops of response constraints. So, it is possible to establish *that SDL processes comply with option to complete if, and only if, they do not contain any response constraints loops.*

Corollary 4.6.1. *From Theorem 4.6.1 and Theorem 4.6.2, it is possible to define that a SDL process is semantically correct, if and only if, the loop of constraints precedence is empty ($L_p = \emptyset$), and the loop of constraints response is empty ($L_r = \emptyset$), in this process.*

In Example 4.6.2 are presented two examples of processes that violate the semantic rules of Corollary 4.6.1.

Example 4.6.2. *The process with $T = \{t_1, t_2, t_3, t_4\}$ and $R =$ precedence(t_1, t_2), precedence(t_2, t_1) is semantically incorrect because $L_p = 2$. The process with $T = \{t_1, t_2, t_3, t_4\}$ and $R =$ response(t_1,t_2), response(t_2,t_3), response(t_3,t_1) is semantically incorrect because $|L_r| = 3$.*

In accordance to Corollary 4.6.1, the semantic correctness of a SDL process is guaranteed just by checking if $Lp = \emptyset$ and $Lr = \emptyset$. If $Lp = \emptyset$ and $Lr = \emptyset$ are true then it is guaranteed set of constraints is valid. In other words, the process modeled by this set of constraints fulfills *option to complete*, *no dead task*, and *proper completion*. Figure 38 shows the procedure for verifying whether a set of constraints is valid. Set of constraints defined by the user is checked in order to identify if there is some *loop of constraints precedence* and *response*. If there is not any *loop of constraints precedence* nor *response* then set of constraints is valid and the process of verifying is finished. If there is some *loop of constraints precedence* or *response* then the user can choose to redefine set of constraints or to finish the process of defining/verifying set of constraints.

4.6.5 Other definitions to SDL processes

The previous subsections presented the conditions required to fulfill the syntax and semantics requirements in SDL frameworks. This section introduces the relations of union, equality, and subset in SDL processes. Then, the relations between syntax and semantics in SDL processes is shown.

Lemma 4.6.1. *Let $A = \{T_A, R_A\}$ and $B = \{T_B, R_B\}$ be two SDL processes. $A \cup B = (T_A \cup T_B, R_A \cup R_B)$.*
Proof: T_A and T_B are sets from tasks and R_A and R_B are sets of constraints. The resulting process from $A \cup B$ must be the union between sets from the same nature. So $A \cup B = (T_A \cup T_B, R_A \cup R_B)$.

Lemma 4.6.2. *Let $A = \{T_A, R_A\}$ and $B = \{T_B, R_B\}$ be two SDL processes. $A = B$ iff $T_A = T_B$ and $R_A = R_B$.*
Proof: T_A and T_B are sets from tasks and R_A and R_B are sets of constraints. The

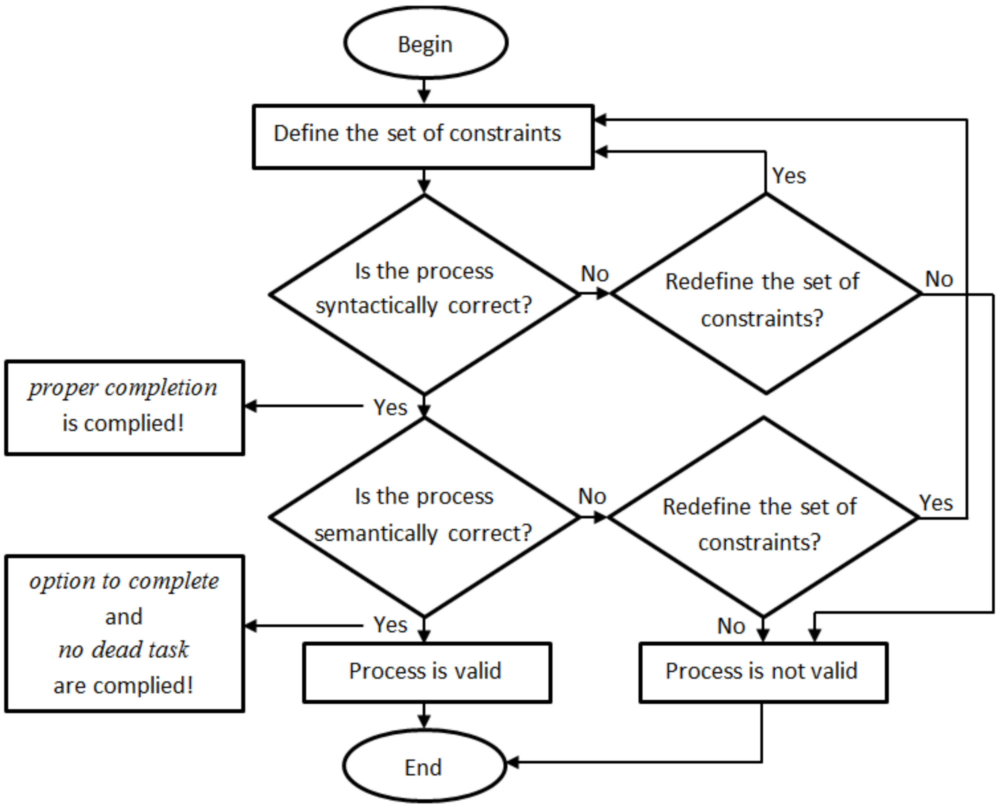

Begin
Define the set of constraints
Is the process syntactically correct?
No
Redefine the set of constraints?
Yes
No
proper completion is complied!
Yes
Is the process semantically correct?
No
Redefine the set of constraints?
Yes
No
option to complete and *no dead task* are complied!
Yes
Process is valid
Process is not valid
End

Figura 38 – Procedure for verifying if a set of constraints is valid.

comparison between two SDL process, to check if they are the same, must be made among the sets from the same nature. So $A = B$ *iff* $T_A = T_B$ *and* $R_A = R_B$.

Lemma 4.6.3. *Let* $A = \{T_A, R_A\}$ *and* $B = \{T_B, R_B\}$ *be two SDL processes.* $A \subseteq B$ *iff* $T_A \subseteq T_B$ *and* $R_A \subseteq R_B$.
Proof: T_A *and* T_B *are sets from tasks and* R_A *and* R_B *are sets of constraints. The comparison between two SDL process, to check if one of them is contained in the other, must be made among the sets from the same nature. So* $A \subseteq B$ *iff* $T_A \subseteq T_B$ *and* $R_A \subseteq R_B$.

Semantics correctness of a SDL process is checked only after syntactic correctness is guaranteed. So if a SDL process is semantically correct then it is syntactically correct. This is Theorem 4.6.3.

Theorem 4.6.3. *Let A be a SDL process. If A is semantically correct then A is syntactically correct.*
Proof: The method to verify whether a set of constraints is valid, presented in Figure 38, imposes that, only after the syntactic correctness has been guaranteed, is the semantic correctness guaranteed.

If two SDL processes are joined and both of them are syntactically correct then their resulting union is a syntactically correct SDL process. This is Theorem 4.6.4.

Theorem 4.6.4. *if A and B are two syntactically correct SDL processes then $A \cup B$ is a syntactically correct SDL process.*

Proof. Hypothesis: A = (T_A, R_A) and B = (T_B, R_B) are two syntactically correct SDL processes and $A \cup B$ = ($T_A \cup T_B$, $R_A \cup R_B$) is a syntactically incorrect SDL process. The hypothesis to be checked is: if $A \cup B$ is syntactically incorrect, then one of the following six cases (C_1 to C_6) is true:

C1: (r is a constraint) $\wedge$ (r $\in R_A \cup R_B$) $\wedge$ (r $\neq$ atleast1(t_i)) $\wedge$ (r $\neq$ response(t_i,t_j)) $\wedge$ (r $\neq$ precedence(t_i,t_j)).
If C1 is true, then P1 is true.
P1: (r $\in R_A$) $\vee$ (r $\in R_B$).
So, if C1 is true, A or B is syntactically incorrect, and this makes hypothesis to be checked be false.

C2: (atleast1(t_i) $\in R_A \cup R_B$) $\wedge$ ($t_i \notin T_A \cup T_B$).
If C2 is true, then P2 is true.
P2: ((atleast1(t_i) $\in R_A$) $\vee$ (atleast1(t_i) $\in R_B$)) $\wedge$ ($t_i \notin T_A$) $\wedge$ ($t_i \notin T_B$).
So, if C2 is true, A or B is syntactically incorrect, and this makes hypothesis to be checked be false.

C3: (response(t_i,t_j) $\in R_A \cup R_B$) $\wedge$ ($t_i \notin T_A \cup T_B$ $\vee$ $t_j \notin T_A \cup T_B$).
If C3 is true, then P3 is true.
P3: ((response(t_i,t_j) $\in R_A$) $\vee$ (response(t_i,t_j) $\in R_B$)) $\wedge$ ((($t_i \notin T_A$) $\wedge$ ($t_i \notin T_B$)) $\vee$ (($t_j \notin T_A$) $\wedge$ ($t_j \notin T_B$))).
So, if C3 is true, A or B is syntactically incorrect, and this makes hypothesis to be checked be false.

C4: response(t_i,t_i) $\in R_A \cup R_B$.
If C4 is true, then P4 is true.
P4: (response(t_i,t_i) $\in R_A$) $\vee$ (response(t_i,t_i) $\in R_B$).
So, if C4 is true, A or B is syntactically incorrect, and this makes hypothesis to be checked be false.

C5: (precedence(t_i,t_j) $\in R_A \cup R_B$) $\wedge$ (($t_i \notin T_A \cup T_B$) $\vee$ ($t_j \notin T_A \cup T_B$)).
If C5 is true, then P5 is true.
P5: ((precedence(t_i,t_j) $\in R_A$) $\vee$ (precedence(t_i,t_j) $\in R_B$)) $\wedge$ (($t_i \notin T_A \wedge t_i \notin T_B$) $\vee$ ($t_j \notin T_A \wedge t_j \notin T_B$)).
So, if C5 is true, A or B is syntactically incorrect, and this makes hypothesis to be checked false.

C6: (precedence(t_i,t_i) $\in R_A \cup R_B$).

If C6 is true, then P6 is true.
P6: ((precedence(t_i,t_i) $\in R_A$) $\vee$ (precedence(t_i,t_i) $\in R_B$)) .
So, if C6 is true, A or B is syntactically incorrect, and this makes hypothesis to be checked be false.

If a syntactic correct SDL process is contained in a semantically correct SDL process then this process is also semantically correct. This is Theorem 4.6.5.

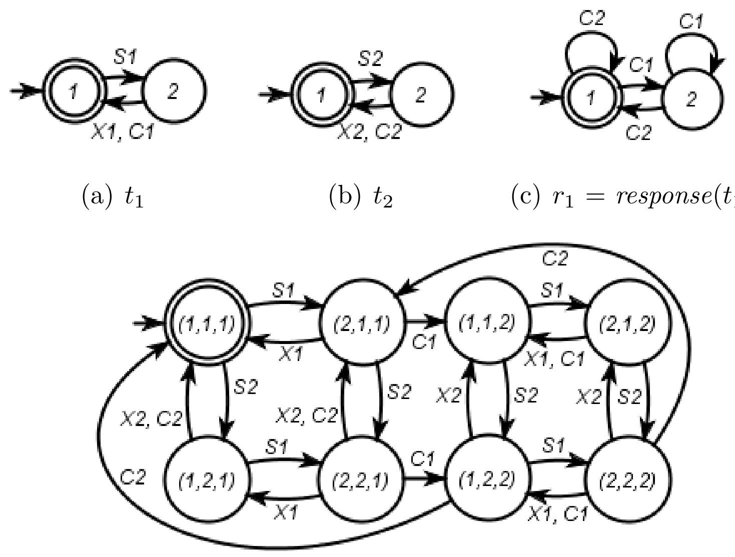
Theorem 4.6.5. *Let A be a semantically correct SDL process. Let B be a syntactically correct SDL process. If $B \subseteq A$ then B is semantically correct.*

Proof. Hypothesis: Let A be a semantically correct SDL process. Let B be a syntactically correct SDL process. $B \subseteq A$, and B is semantically incorrect.
If B is semantically incorrect then there is a loop of precedence constraints (L_p) in B, or there is a loop of response constraints (L_r) in B. If there is a loop L_p in B or there is a loop L_r in B, then there is a loop L_p in A, or there is a loop L_r in A, since $B \subseteq A$. But if there is a loop L_p in A or there is a loop L_r in A, then A is semantically incorrect, and the Hypothesis is false. With the Hypothesis being false, Theorem 4.6.5 is proven to be true.

4.6.6 Enabled and pendent events in SDL

In SDL, a marker is a sequence of numbers where each number represents the current state of one automaton in the process. For example, Figures 39(a), 39(b) and 39(c) show the automata of the tasks t_1 and t_2 and constraint r_1 = *response*(t_1,t_2), each state of these automata is identified by a number, so the marker for this set of tasks and constraints can be established as M = (i,j,k) where i is the number of the current state of t_1, j is the number of the current state of t_2, and k is the current state of r_1. Figures 39(d) shows the synchronous product of t_1, t_2 and r_1 and the marker at each state of the automaton.

The set of enabled events for each marking is calculated by the synchronous product method. In automaton in Figure 39(d), in the marking M = (1,1,1), s_1 and s_2 are enabled. In marking M = (2,1,1), c_1, x_1 and s_2 are enabled, and so on. The set of pendent events for each marking is calculated as demonstrated in the following. To each state of the automaton of a task or constraint is associated a set P of pendent events, the Table 9 shows the set P for every state of the automata of task and constraints *atleast1*(t_i), *precedence*(t_i,t_j) and *response*(t_i,t_j) presented in Figures 30, 31(a), 31(g) and 31(h). C(P) is the collection of all the sets P of all the automata at each marking. Table 8 presents the three cases that must be checked to identify what are the pendent events of a task. So, the method to identify a pendent event of a task just consists in checking which of these three cases is true.



(a) t_1 (b) t_2 (c) r_1 = *response*(t_1,t_2)

(d) $t_1||t_2||r_1$

Figura 39 – Markers for the synchronous product of t_1, t_2 and r_1

Tabela 8 – Three cases to check the pendent events in a task

Case 1: $\{c_i\} \in C(\mathrm{P})$	pendent event = c_i
Case 2: $\{c_i\} \notin C(\mathrm{P})$ and $\{c_i, x_i\} \in C(\mathrm{P})$	pendent event = c_i or x_i
Case 3: $\{c_i\} \notin C(\mathrm{P})$ and $\{c_i, x_i\} \notin C(\mathrm{P})$	no event is pendent

Tabela 9 – Pendent events in automata *task*, *atleast1*(t_i), *precedence*(t_i,t_j) and *response*(t_i,t_j)

Automaton	state 1	state 2
task	P = $\emptyset$	P = $\{c, x\}$
atleast1(t_i)	P = $\{c_i\}$	P = $\emptyset$
precedence(t_i, t_j)	P = $\emptyset$	P = $\emptyset$
response(t_i, t_j)	P = $\emptyset$	P = $\{c_j\}$

Table 10 shows the pendent events for automaton $t_1||t_2||r_1$ in Figure 39(d).

4.6.7 An example

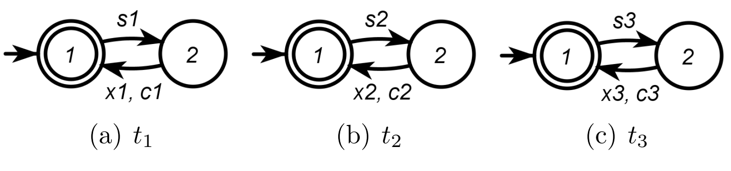
This section presents an example of process design and run using an SDL framework.

At design time, the user designs process EXAMPLE = (T, R), such that T = $\{t_1, t_2, t_3\}$ and R = $\{r_1$ = *atleast1*(t_2), r_2 = *precedence*(t_1, t_2), r_3 =*response*(t_3, t_1)$\}$. After the user has designed the process, it must be checked applying procedure for verifying if a set of constraints is valid, shown in Figure 38. First, the procedure in Figure 38 checks process

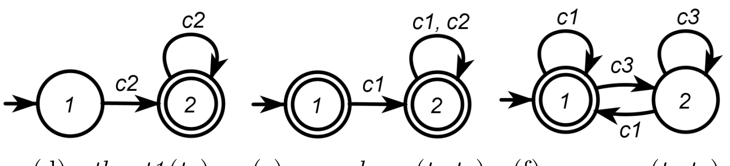
Tabela 10 – Pendent events for automaton $t_1||t_2||r_1$.

Marking	C(P)	Pendent events
1,1,1	$\{\emptyset, \emptyset, \emptyset\}$	none
1,2,1	$\{\emptyset, \{c_2, x_2\}, \emptyset\}$	c_2 or x_2
2,1,1	$\{\{c_1, x_1\}, \emptyset, \emptyset\}$	c_1 or x_1
2,2,1	$\{\{c_1, x_1\}, \{c_2, x_2\}, \emptyset\}$	(c_1 or x_1) and (c_2 or x_2)
1,1,2	$\{\emptyset, \emptyset, \{c_2\}\}$	c_2
1,2,2	$\{\emptyset, \{c_2, x_2\}, \{c_2\}\}$	c_2
2,1,2	$\{\{c_1, x_1\}, \emptyset, \{c_2\}\}$	(c_1 or x_1) and c_2
2,2,2	$\{\{c_1, x_1\}, \{c_2, x_2\}, \{c_2\}\}$	(c_1 or x_1) and c_2

compliance with the rules for a syntactically correct process. These rules are shown in Definition 4.6.1. Since the process is compliant with the rules stated in Definition 4.6.1, it is syntactically correct. After that, the procedure in Figure 38 checks process compliance with the rules for a semantically correct process. These rules are stated in Corollary 4.6.1. Since the process is compliant with the rules stated in Corollary 4.6.1, it is semantically correct. Since the process is syntactically and semantically correct, it is a valid SDL process. The set of tasks and constraints comprising process P is the output of design time. Figure 40 exhibits the sets of tasks and constraints for process EXAMPLE.



(a) t_1 (b) t_2 (c) t_3



(d) *atleast1*(t_2) (e) *precedence*(t_1,t_2) (f) *response*(t_3,t_1)

Figura 40 – Tasks and constraints of the process EXAMPLE

At run time, the SDL framework takes sets of valid tasks and constraints, generated during design time, and compiles Table 11. Table 11 displays the sets of pending events (P) to the automata of process EXAMPLE. Table 11 is prepared from the templates shown in Table 9.

The next step is providing the initial marker of the running process. The markers are defined by the number of the state for each process automata. The order of the automata that defines the markers is t_1, t_2, t_3, r_1, r_2, r_3. So, the initial marker in process EXAMPLE is M_1 = *111111*.

The next step is providing sets of enabled and pending events to M_1.

Tabela 11 – Sets of pendents events (P) to each construct of the process EXAMPLE

Construct	Automaton	state 1	state 2
t_1	task	$\mathrm{P}_{t_1} = \emptyset$	$\mathrm{P}_{t_1} = \{c_1, x_1\}$
t_2	task	$\mathrm{P}_{t_2} = \emptyset$	$\mathrm{P}_{t_2} = \{c_2, x_2\}$
t_3	task	$\mathrm{P}_{t_3} = \emptyset$	$\mathrm{P}_{t_3} = \{c_3, x_3\}$
r_1	*atleast1*(t_2)	$\mathrm{P}_{r_1} = \{c_2\}$	$\mathrm{P}_{r_1} = \emptyset$
r_2	*precedence*(t_1, t_2)	$\mathrm{P}_{r_2} = \emptyset$	$\mathrm{P}_{r_2} = \emptyset$
r_3	*response*(t_3, t_1)	$\mathrm{P}_{r_3} = \emptyset$	$\mathrm{P}_{r_3} = \{c_1\}$

The set of enabled events, for each marker, is calculated applying a synchronous product operation. The synchronous product operation is explained in Section 4.5.1. For marker M_1, the set of enabled events is E = $\{s_1, s_2, s_3\}$.

The set of pending events, for every marker, is calculated applying the method set out in Section 4.6.6. So, the collection of sets of pending events ($C(P)$) for every marker must be calculated for each marker in process EXAMPLE, $(C(P)) = \{P_{t_1}, P_{t_2}, P_{t_3}, P_{r_1}, P_{r_2}, P_{r_3}\}$. From $(C(P))$, in each marker, the true case out of the three presented in Table 8 must be determined.

So, in M_1, $C(P) = \{\emptyset, \emptyset, \emptyset, \{c_2\}, \emptyset, \emptyset\}$, and, consequently, the set of pending events is P = $\{c_2\}$.

From M_1, the user chooses to execute event s_3. Now, the *method to update the current state in the automata* takes event executed s_3 and updates the state in each process automata. Section 4.6.2 shows the *method to update the current state in the automata.* This update causes automaton t_3 to change from *state 1* to *state 2.* Automata t_1, t_2, r_1, r_2, and r_3 do not change their respective states.

Since the state of automaton t_3 has changed, a new marker is reached – marker M_2. So, M_2 = *112111.* In M_2, the set of pending events is $E = \{s_1, s_2, c_3, x_3\}$. In M_2, $C(P) = \{\emptyset, \emptyset, \{c_3, x_3\}, \{c_2\}, \emptyset, \emptyset\}$. Consequently, in M_2, set of pending events is $P = \{c_2, c_3 \text{ ou } x_3\}$.

From M_2, the user chooses to execute event s_1. Now, the method to update the current state in the automata takes the event executed s_1 and updates the state in each of the process automata. This updates changes automaton t_1 from the *state 1* to *state 2.* Automata t_2, t_3, r_1, r_2, and r_3 do not change their respective states.

Since the state of automaton t_1 has changed, a new marker is reached – marker M_3. So, M_3 = *212111.* In M_3, set of enabled events is $E = \{c_1, x_1, s_2, c_3, x_3\}$. In M_3, $C(P) = \{\{c_1, x_1\}, \emptyset, \{c_3, x_3\}, \{c_2\}, \emptyset, \emptyset\}$. Consequently, in M_3, the set of pending events is $P = \{c_1 \text{ ou } x_1, c_2, c_3 \text{ ou } x_3\}$.

From M_3, the user chooses to execute event c_3. Now, the *method to update the current state* *in the automata* takes the event executed c_3 and updates the state in each of the process automata. This update changes automaton t_3 from *state 2* to *state 1* and automaton r_3 changes from *state 1* to *state 2*. Automata t_1, t_2, r_1, and r_2 do not change their respective states.

Since the state of automata t_3 and r_3 has changed, a new marker is reached - marker M_4. So, $M_4 =$ *211112*. In M_4, the set of enabled events is $E = \{c_1, x_1, s_2, s_3\}$. In M_4, $C(P) = \{\{c_1, x_1\}, \emptyset, \emptyset, \{c_2\}, \emptyset, \{c_1\}\}$. Consequently, in M_4, set of pending events is $P = \{c_1, c_2\}$.

From M_4, the user chooses to execute event s_2. Now, the *method to update the current state in the automata* takes event executed s_2 and updates the state in each of the process automata. This update changes automaton t_2 from *state 1* to *state 2*. Automata t_1, t_3, r_1, r_2, and r_3 do not change their respective states.

Since the state of the automaton t_2 has changed, a new marker is reached – marker M_5. So, $M_5 =$ *221112*. In M_5, set of enabled is $E = \{c_1, x_1, c_2, x_2, s_3\}$. In M_5, $C(P) = \{\{c_1, x_1\}, \{c_2, x_2\}, \emptyset, \{c_2\}, \emptyset, \{c_1\}\}$. Consequently, in M_5, the set of pending events is $P = \{c_1, c_2\}$.

From M_5, the user chooses to execute event c_1. Now, the *method to update the current state in the automata* takes event executed c_1 and updates the state in each of the process automata. This update changes automata t_1 from *state 2* to *state 1*, r_2 changes from *state 1* to *state 2*, and r_3 changes from *state 2* to *state 1*. Automata t_2, t_3, r_1, and r_2 do not change their respective states.

Since the state of automata t_1, r_2 and r_3 has changed, a new marker is reached - marker M_6. So, $M_6 =$ *121121*. In M_6, the set of enabled events is $E = \{s_1, c_2, x_2, s_3\}$. In M_6, $C(P) = \{\{c_1, x_1\}, \{c_2, x_2\}, \emptyset, \{c_2\}, \emptyset, \emptyset\}$. Consequently, in M_6, the set of pending events is $P = \{c_2\}$.

From M_6, the user chooses to execute event c_2. Now, the *method to update the current state in the automata* takes event executed c_2 and updates the state in each of the process automata. This update changes automata t_2 from *state 2* to *state 1*, and r_1 changes from *state 1* to *state 2*. Automata t_1, t_3, r_2, r_3 do not change their respective states.

Since the state of automata t_2, and r_1 has changed, a new marker is reached - marker M_7. So, $M_7 =$ *111221*. In M_7, set of enabled is $E = \{s_1, s_2, s_3\}$. In M_7, $C(P) = \{\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\}$. Consequently, in M_7, set of pending events is $P = \emptyset$.

In M_7 there are no pending events. So, in M_7, the user can choose to continue or finish the process.

Figure 41 shows markers M_1 to M_7 that are reached through the sequence of events $s_3.s_1.c_3.s_2.c_1.c_2$.

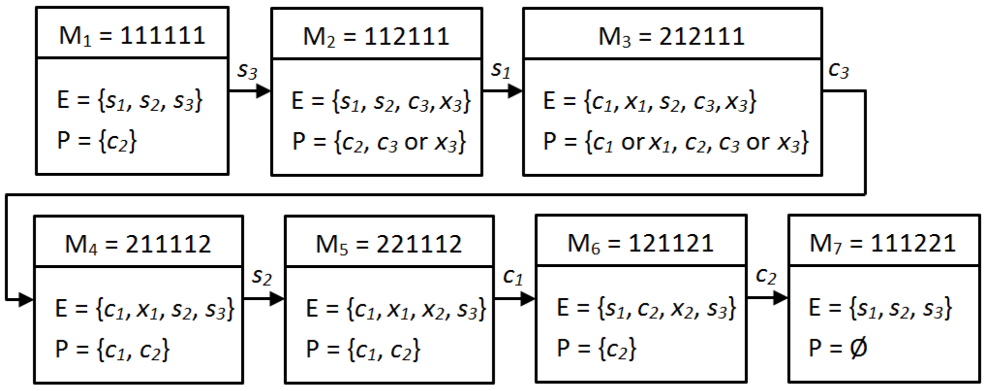


Figura 41 – Markers that are reached after the sequence $s_3.s_1.c_3.s_2.c_1.c_2$ is executed .

4.7 Conclusion

This paper introduces Simple Declarative Language (SDL), a conceptual framework for modeling constraint based processes.

SDL frameworks provide a single task model that is used in every process modeled. This task model is the same as in the SCT approach. SDL frameworks provide a set of only three constraints: existence, precedence, and response. These models of constraints are the same as in the SCT approach.

SDL provides controllable event start, complete and cancel tasks. SDL does not provide uncontrollable events.

At design time, the SDL framework provides a method to check SDL language syntax and semantics. This method checks compliance of sets of tasks and constraints with the applicable syntactic and semantics rules defined for the SDL language. Semantic rules in SDL frameworks relate to soundness. If an SDL process is semantically correct, then it is compliant with the respective soundness requirements. The output of design time is only a set of tasks and constraints. No sequence of events is calculated in design time.

The syntax rules in SDL frameworks are described in section 4.6.3. If the set of tasks and constraints is compliant with the conditions described in section 4.6.3, then the process is syntactically correct, otherwise the process is not syntactically correct.

Section 4.6.4 describes the semantics rules used in SDL frameworks. If the set of tasks and constraints is compliant with the conditions described in section 4.6.4. then the process is semantically correct, otherwise the process is not semantically correct.

At run time, SDL frameworks provide four methods. *Valid set of tasks and constraints* is the input to *Method to calculate pending events* and *Method synchronous product. Method to calculate pending events* takes the *valid set of tasks and constraints* to calculate *set of pending events* in the process. *Method synchronous product* takes *set of tasks and constraints* to calculate *set of enabled events* for the process. From the *set of enabled events*, the user chooses to execute an event. When the user chooses to execute an event, *Method to update sequence of executed events* updates its outputs: *Sequence of events executed* and *Last event executed*.

The main limitation of SDL frameworks is its language. The SDL language provides only three constraints. This can be very restrictive in cases where, for example, it is necessary to model processes with constraints that provide for the exclusion of events.

4.7.1 Future work

The next step in developing SDL is providing it with *constraint exclude*. This will increase the power of SDL enabling modeling a greater number of constraint based processes.

5 A conceptual framework to select variants from constraint based processes

Abstract

Different process models may be used in varying contexts of the same application if components in these processes can be partially modified. These process models can be also set in accordance with attributes related to different circumstances. Variability is the type of flexibility that permits configuring a process model to address specific circumstances. Process variability may be a requirement in different domains when processes need to be handled as a function of different business process contexts driving the need for different process variants. Designing and implementing standardized business processes for each context of the real world becomes too expensive for businesses. So, there is a high level of interest in gathering common process knowledge for deployment as a reference process model, and, consequently, derive all variants in alignment with each different application context. Most of the studies on selection of business process variants focus on imperative languages. There are few studies about selection of variants with declarative languages. Therefore, the objective of this paper is to propose a business process variant selection framework modeled using Simple Declarative Language (SDL)- a conceptual framework for modeling constraints based processes.

Keywords: Variability, selection of variants, configurable process model, imperative languages, declarative languages.

5.1 Introduction

Different process models may be used in varying contexts of the same application if components in these processes can be partially modified (REICHERT; WEBER, 2012a). These process models can be also set in accordance with attributes related to different circumstances (REICHERT; HALLERBACH; BAUER, 2015). These conditions allow some types of process to be reused in different contexts or circumstances as long as the necessary changes to some of their components are carried out (ROSA et al., 2013). Reusing a process model in different contexts can result in a wide range of related process model variants belonging to the same process family (MILANI et al., 2016). These process variants are connected to the same business objectives and have several points in common (ROSA et al., 2013). But there are also differences due specific conditions found in each context, for example, some activities may be required for a given context, but, may be entirely unnecessary for other contexts (SCHUNSELAAR et al., 2014).

Variability is the type of flexibility that permits configuring a process model to address specific circumstances (MECHREZ; REINHARTZ-BERGER, 2014). Process variability may be a requirement in different domains when processes need to be handled as a function of different business process contexts driving the need for different process variants (BERGER et al., 2013; HUANG et al., 2013). Process variants are usually derived from the same process model with the actual sequence of actions varying for each variant (AALST, 2013).

At least four aspects can generate process variants: products and services, regulations and laws, type of clients, and time(REICHERT; WEBER, 2012a). Product and service variability are required because there may be effective product variants in the same business. Differences in regulations in different countries and regions can also drive the need for different process variants in the same company. Variability might be also required in addressing different types of customers (premium or standard, for example) as well as due to temporal differences (seasonal changes, for example). The actual variant can be defined while the process is being executed, but the configurable process model from which each variant is derived must be known previously. Health-care processes for emergency patient treatment are examples of process variability (LENZ; PELEG; REICHERT, 2012). Before each patient is treated, his/her general condition is evaluated resulting in a scenario used in defining which actions must be executed, and which must not, from a configurable process model standpoint, and the resulting behavior is a process variant.

Designing and implementing standardized business processes for each context of the real world becomes too expensive for businesses (AYORA et al., 2012). This results in a high level of interest in gathering common process knowledge to use as a reference process model, and, consequently, derive all variants in aligning with each context of application (AYORA et al., 2013b). Thus, an approach to capture and set the variability in a process model is needed. This approach must be able to represent a family of process variants in a compact, reusable, and maintainable way, as well as allowing configuring a process family in such a way that every process variant represents, correctly, the requirements of a specific environment for the application (AYORA et al., 2015).

The adoption of process-aware information systems (PAISs) (REICHERT; WEBER, 2012b) such as workflow management systems (DUMAS et al., 2013; AALST; WESKE, 2013), enterprise resource planning systems (SHAUL; TAUBER, 2013; MONK; WAGNER, 2012), or case management systems (MOTAHARI-NEZHAD; SWENSON, 2013) is increasing due to the high variability in business processes (e.g., sales processes may vary depending on the respective products and countries). Dealing with process families properly constitutes the main challenge in reducing development and maintenance costs in large process repositories (DIJKMAN; ROSA; REIJERS, 2012). Designing and implementing each process variant from scratch and maintaining it separately would be inefficient and costly for companies (TEALEB; AWAD; GALAL-EDEEN, 2014; YAN; DIJKMAN; GREFEN, 2012).

Thus, there is a great interest in capturing common process knowledge only once and re-using it as reference for process models, e.g., ITIL in IT service management (IDEN; EIKEBROKK, 2013; TRUSSON; DOHERTY; HISLOP, 2014; MARRONE et al., 2014), reference processes in SAP ERP systems (LORENC; SZKODA, 2015; GÖTZFRIED et al., 2013; YANG; SEN; PING, 2013), or medical guidelines (HERZBERG; KIRCHNER; WESKE, 2014; ROJAS et al., 2016). Even though these proposals promote the reuse of common process knowledge, typically, they lack comprehensive support in explicitly describing variations (AYORA et al., 2013a).

More specifically, a business process variant selection framework requires sets of procedures and data that allow merging processes from the same domain application in order to facilitate process management for them (AYORA et al., 2015; YAN; DIJKMAN; GREFEN, 2012).

The data structure where processes from the same application domain are merged is called Configurable Process Model (ASSY; GAALOUL, 2015; JIMÉNEZ-RAMÍREZ et al., 2015; SHARMA; RAO et al., 2015). Configurable Process Model is essential part in a variant selection procedure (DÖHRING; REIJERS; SMIRNOV, 2014). Each process that is merged in the Configurable Process Model is a variant (COGNINI et al., 2014; ASADI et al., 2014; SCHUNSELAAR et al., 2014b). Context-specific variants can be selected from the Configurable Process Model (MURGUZUR et al., 2014; ASSY; GAALOUL, 2014; TEALEB; AWAD; GALAL-EDEEN, 2015; HACHICHA et al., 2016).

Variants must be syntactically and semantically correct before they can be merged into a Configurable Process Model, and each variant selected from the Configurable Process Model must be syntactically and semantically correct (ROSA, 2009; ROSA et al., 2013; TORRES et al., 2012). This requires defining the syntax and semantics rules of the language that is used to model the process variants.

In order to facilitate the variant selection procedure, some types of support for selection of process variants can be implemented (REICHERT; WEBER, 2012a; ROSA et al., 2013; AYORA et al., 2012). There are, at least, four support techniques to help with selecting of process variants: questionnaire-driven configurations, feature models, goal models and decision tables (AYORA et al., 2012). This paper concentrates on questionnaire-driven configurations (also called questionnaire-based framework). In questionnaire-based frameworks, the questionnaire acts as user interface enabling domain-specific representations of configuration decisions. The questionnaire model comprises a set of domain facts corresponding to the answers of a set of questions in natural language (REICHERT; WEBER, 2012a). Each domain fact corresponds to a Boolean variable representing a particular feature of the domain. These feature may be enabled or disabled depending on the given application context (ROSA et al., 2013).

In recent years, a number proposals have been made to deal with selection of variants from process families. In the BP management field, model-driven techniques provide diverse solutions for managing process variants, i.e. for modeling, configuring, executing, and monitoring a process family (ZHANG; HAN; OUYANG, 2014; ASSY; CHAN; GAALOUL, 2015; YONGSIRIWIT; ASSY; GAALOUL, 2016). However, most of the studies on selecting business process variants, developed to date, concentrate on imperative languages (AYORA et al., 2015). These studies have introduced frameworks intended to support procedures in making and selecting variants from configurable process models by using processes that are modeled using imperative languages. There are few studies about selecting variants using declarative languages (SCHUNSELAAR et al., 2012a) There is a dearth of studies on frameworks intended to generate configurable process models in which the variants are modeled using declarative languages.

Thus, this paper's main objective is to propose a framework to merge and select variants from the configurable process model in which the processes are modeled through a declarative language. In this framework, the variants are created syntactically and semantically correct and, when a variant is selected, its syntactic and semantic correctness is preserved. The variant selection process is supported by a questionnaire based approach. Figure 42 presents the simplified diagram for this framework. At design time, users are provided with *User support framework at design time*. This framework provides all the necessary tools users can apply *User support framework at configure time*. At configure time, *User support framework at configure time* all necessary tools are provided for users to generate variants (an SDL process). At run time, *User support framework at run time* all necessary tools for users to run the variant are provided.

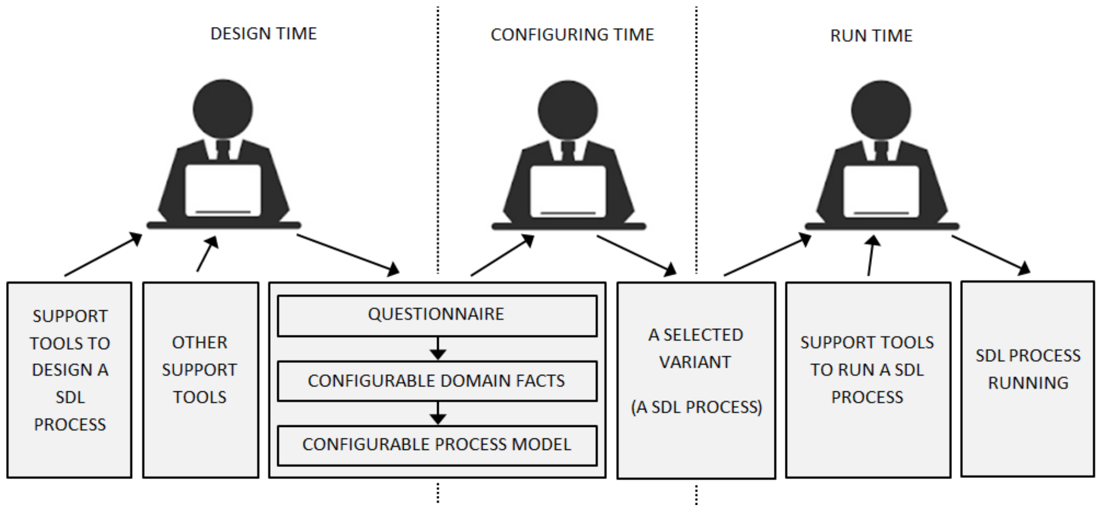

DESIGN TIME
CONFIGURING TIME
RUN TIME
SUPPORT TOOLS TO DESIGN A SDL PROCESS
OTHER SUPPORT TOOLS
QUESTIONNAIRE
CONFIGURABLE DOMAIN FACTS
CONFIGURABLE PROCESS MODEL
A SELECTED VARIANT
(A SDL PROCESS)
SUPPORT TOOLS TO RUN A SDL PROCESS
SDL PROCESS RUNNING

Figura 42 – Simplified representation of the framework proposed in this work

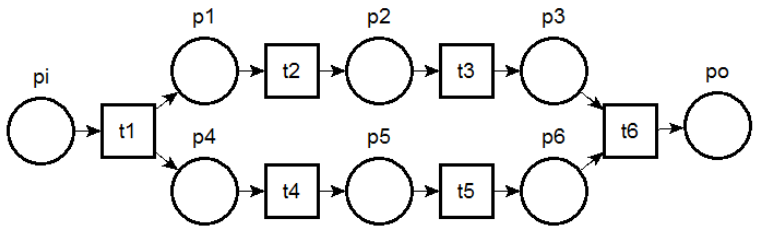
5.2 Configurable process model

If there is a set of processes related to the same knowledge domain and, each of these processes is syntactically and semantically correct, then this set of processes is called a family of process. The processes in a given family can be merged into larger processes called configurable process model (REICHERT; HALLERBACH; BAUER, 2015; SCHUNSELAAR et al., 2014a; BUIJS; DONGEN; AALST, 2013; AYORA et al., 2013b). From a configurable process model, procedures and methods can be defined enabling users to select one of the processes in the process family. Each process selected from the configurable process model is called a variant. The configurable process model must be also syntactically and semantically correct in order to guarantee that every variant remains syntactically and semantically correct after the selection procedure has been performed (AYORA et al., 2015). An advantage of configurable process models is avoiding redundant work (ROSA et al., 2013). For example, if some change must be made in a given task, then all the processes in the family that share that task will be modified with a single procedure, waiving the need to make the same change in each process individually.

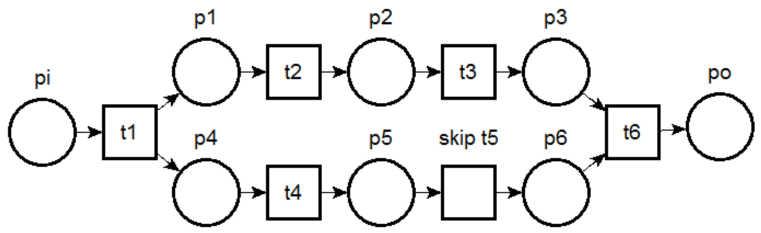
Figure 43(a) to 43(d) presents four process variants for a hypothetical domain. Variant 1 has 6 tasks (t_1, t_2, t_3, t_4, t_5, t_6). In variant 2, t_5 is hidden, i.e. t_5 is skipped by the transition *skip* t_5. In variant 3, t_2 is hidden, i.e. t_2 is skipped by the transition *skip* t_2. In variant 4, t_2 and t_5 are hidden, i.e. t_2 and t_5 are skipped by the transitions *skip* t_2 and *skip* t_5. A *skip transition* is a *silent transition*. A silent transition allows firing the tokens in their incoming points but, differently from a transition that represents a task, it does not produce any effect on the process. The four variants are merged in a single process shown in Figure Figure 43(e). The process in Figure Figure 43(e) is a free-choice workflow net, thus it is syntactically and semantically correct. So, the process in Figure 43(e) can be used as a configurable process model.

In a configurable process model, there *variation point* is a limited region of the process that offers a limited amount of process configuration options. Each option at a *variation point* is also called a *process fact* (ROSA, 2009). There are two variation points in the configurable process model shown in Figure 43(e): *variation point 1* and *variation point 2*. The *variation point 1* is the region comprised between points p_1 and p_2, and *variation point 2* is the region comprised between points p_5 and p_6.

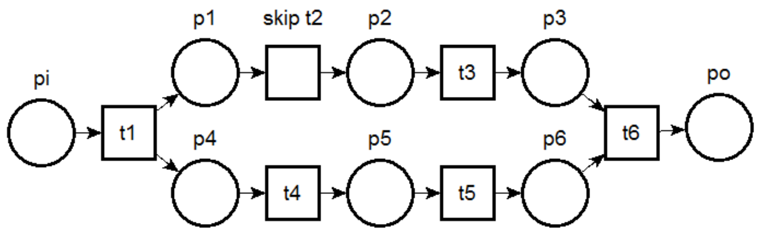
In *variation point 1* there are two *process facts*: *process fact 1* and *process fact 2*. In *variation point 2* there are two *process facts*: *process fact 3* and *process fact 4*. Table 12 shows the variation points and process facts for the configurable process model in Figure 43(e). The *process fact 1* is equivalent to t_2 is *allowed*. The *process fact 2* is equivalent to t_2 is *hidden*. The *process fact 3* is equivalent to t_5 is *allowed*. The *process fact 4* is equivalent to t_5 is *hidden*.



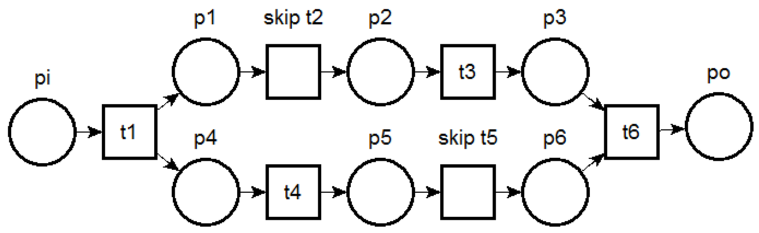
(a) Variant 1



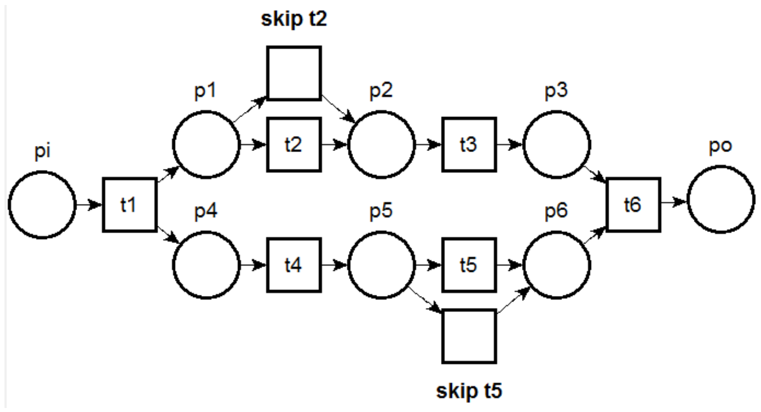
(b) Variant 2



(c) Variant 3



(d) Variant 4



(e) Merge of the Variant 1, Variant 2, Variant 3 and Variant 4

Figura 43 – Four variants and their mergers

Tabela 12 – Variation points and process facts to the configurable process model in Figure 43(e)

vp_1:	$pf_1 \Leftrightarrow t_2$ is *allowed*
	$pf_2 \Leftrightarrow t_2$ is *hidden*
vp_2:	$pf_3 \Leftrightarrow t_5$ is *allowed*
	$pf_4 \Leftrightarrow t_5$ is *hidden*

5.3 Domain facts and questionnaire

The process fact choosing activity when selecting variants for a configurable process model may not be so intuitive, and, therefore, may be complex and confusing for users. In order to render the variant selection process from a configurable process model more intuitive and easier, each *process fact* is bound to a set of *domain facts*. A *domain fact* is a description, in natural language, of a particular feature in the application domain of the configurable process model (REICHERT; WEBER, 2012a; ROSA, 2009).

For example, for a configurable process model in the health-care domain, the type of the medical examination could be defined by two domain facts: *Emergency Medical Examination* and *Standard Medical Examination* (REICHERT; WEBER, 2012a). Choosing one of those domain facts would define the process facts (options) that should be chosen in order to perform the appropriate actions in the respective process. Users do not need to know the details of the process structure to be able to select the correct variant. They just need to know the domain of the process (in this case, health-care) to select the correct variant.

For the configurable process model in Figure 43(e) 4 hypothetical domain facts have been defined. Each of these hypothetical domain facts corresponds to a hypothetical features as shown in Table 13. Each process fact must be bound to a logical expression defined through domain facts. Table 14 shows the process facts of the configurable process model in Figure 43(e) with their logical expressions. For example, if *domain fact 1* and *domain fact 3* are *true* then the *process fact 1* is *true* and consequently t_2 is *allowed* is *true*.

Tabela 13 – Correspondence between domain facts and features

$domainfact$	correspondent feature
f_1	*feature 1*
f_2	*feature 2*
f_3	*feature 3*
f_4	*feature 4*

Tabela 14 – Process facts from Figure 43(e) with their logical expressions

process fact 1	$\Leftrightarrow$	$f_1 \wedge f_3$
process fact 2	$\Leftrightarrow$	$f_2 \wedge f_3$
process fact 3	$\Leftrightarrow$	$f_1 \wedge f_4$
process fact 4	$\Leftrightarrow$	$f_3 \wedge f_4$

5.3.1 Grouping domain facts into questions

The *domain facts* can be grouped in individual questions in a questionnaire. This is known as the questionnaire-based approach. The questionnaire-based approach is one of several support approaches applicable in selecting variants for a business process (AYORA et al., 2015). The questions are also stated in natural language. Thus, by answering a questionnaire, users can select the pertinent process facts, and, accordingly, the appropriate process variant, from the configurable process model. This approach is very useful since it provides a more user friendly interface when selecting a process variant (REICHERT; WEBER, 2012a; ROSA, 2009).

Figure 44 shows a partial example of a variant selection questionnaire for use with configurable process model in the health-care domain. *Fact 1* (*Emergency Medical Examination*) and *fact 2* (*Standard Medical Examination*) are bound to question *Q1* (*Shall a standard or an emergency medical examination be handled?*). *Fact 1*, *fact 2* and question *Q1* support users in defining the type of the medical examination to be performed on the patient. *Fact 3* (*Appointment Required*) and *fact 4* (*Simple Registration Sufficient*) are bound to question *Q2* (*Does an appointment for the standard medical examination have to be arranged?*). *Fact 3*, *fact 4* and question *Q1* support users in defining whether an appointment is required for the standard medical examination.

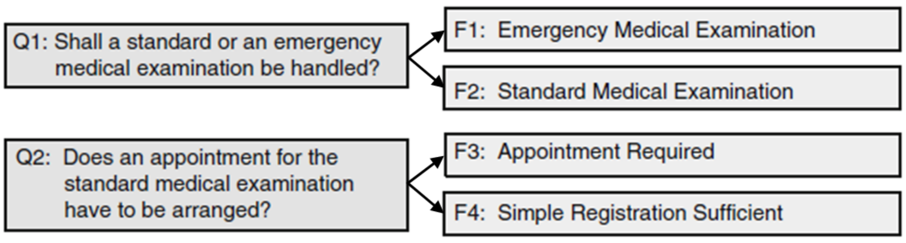

Q1: Shall a standard or an emergency medical examination be handled?
F1: Emergency Medical Examination
F2: Standard Medical Examination
Q2: Does an appointment for the standard medical examination have to be arranged?
F3: Appointment Required
F4: Simple Registration Sufficient

Figura 44 – Part of a questionnaire for selection of variants in the health-care domain

Binding domain facts and questions provide users with a friendly interface. The questions help users to think about the context in which the facts (options) are inserted. For example, *Q1* helps users to establish whether the context of the facts to be chosen (f_1 and f_2) is related to the type of examination, whereas *Q2* helps users to establish whether the context of the facts to be chosen (f_3 and f_4) is related to making (or not) an appointment for a Standard Medical Examination. For the configurable process model in Figure 43(e), the 4 domain facts are grouped into 2 questions as shown in Figure 45.

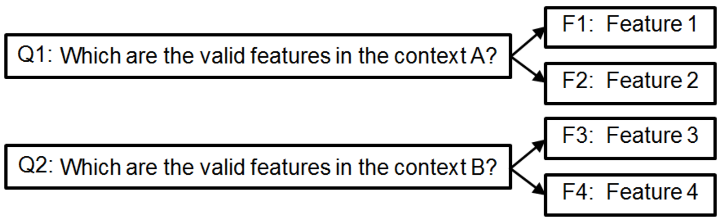

Q1: Which are the valid features in the context A?
F1: Feature 1
F2: Feature 2
Q2: Which are the valid features in the context B?
F3: Feature 3
F4: Feature 4

Figura 45 – Domain facts from Figure 43(e) grouped in two questions

5.3.2 Domain fact constraints and order dependencies

After defining the domain facts, there may be constraints that must be imposed on them (REICHERT; WEBER, 2012a; ROSA, 2009; AYORA et al., 2015). For example, for domain facts *Emergency Medical Examination* and *Standard Medical Examination*, mentioned previously, a constraint to be imposed is that they are mutually exclusive. If these domain facts are respectively f_i and f_j then that constraint can be represented by the logical expression $f_i \oplus f_j$. In the configurable process model in Figure 43(e) 4 domain fact constraints are defined. These constraints are shown in Table 15. The constraints in Table 15 avoid non-valid variants from being selected from the configurable process model in Figure 43(e).

Tabela 15 – Constraints avoid a non-valid variant from being selected from Figure 43(e)

description of the constraint	logical expression
exclusively f_1 or exclusively f_2 must be *true*	$f_1 \oplus f_2$
exclusively f_3 or exclusively f_4 must be *true*	$f_3 \oplus f_4$

Domain facts may also require order dependency among them. There may be some requirement between two facts such that one of them should always be performed before the other. For example, for the domain facts in Figure 44, the type of medical examination to be performed must be established (f_1 and f_2) before defining how to arrange the Standard Medical Examination (f_3 and f_4). So f_1 and f_2 must be set before f_3 and f_4. So f_1 and f_2 must precede f_3 and f_4. Accordingly, since questions Q1 and Q2 inherit the order dependency from their facts, Q1 must precede Q2. The hypothetical application domain of the configurable process model in Figure 43(e) requires that f_1 and f_2 be the first ones to be set, and that f_3 and f_4 be set before f_5 and f_6. Once the questions inherit the order dependency from their facts, the order in which the questions must be answered is Q1, Q2.

5.4 Selection of Variants with Simple Declarative Language (SVSDL)

This section presents Selection of Variants with Simple Declarative Language (SVSDL). SVSDL is a conceptual framework to variants selection from constraint based processes modeled by Simple Declarative Language (SDL). SVSDL covers three different times: design time, configuring time, and run time. At design time, SVSDL provides user with *User support framework at design time.* At configure time, SVSDL provides user with *User support framework at configure time.* At run time, SVSDL provides user with *User support framework at run time. User support framework at run time* is the same one provided by SDL framework at run time(SCHAIDT; SANTOS, 2017b). Figure 57 and 58 present these frameworks.

In the next section, *User support framework at design time* and *User support framework at configure time* are presented in more details. Since *User support framework at run time* is the same one provided by the SDL framework, this section only presents a brief description of this framework.

5.4.1 Design time

At design time, SVSDL provides users with *User support framework at design time. User support framework at design time* makes *User support framework at configure time.* Next, the *User support framework at design time* methods, as well as the fundamentals they are based on, are introduced.

5.4.1.1 Method to define Function

This method gets no input from other methods. The output from this method is *Function. Function* can have one of two values: *Exactly.one* or *Atleast.one*. The value of *Function* defines the way the variants are selected in SVSDL. Figure 48 shows the *Method to define Function.* This method is very simple, consisting in merely asking the user attribute of one of the two values (*Exactly.one* or *Atleast.one*) for *Function.*

If *Function* = *Exactly.one* then, in configure time, users can only select one variant from *Configurable Process Model.* After one variant has been selected from *Configurable Process Model*, the procedure to select variants is finished. If *Function* = *Atleast.one* then, in configure time, users can select more than one variant from *Configurable Process Model.* After one variant has been selected from *Configurable Process Model*, the procedure to select variants is finished only if users choose this. In fact, if *Function* = *Atleast.one* then the "final variant"to be executed in run time is the inion of the variants selected by users during configure time. This matter is discussed in more detail in section 5.4.1.4.

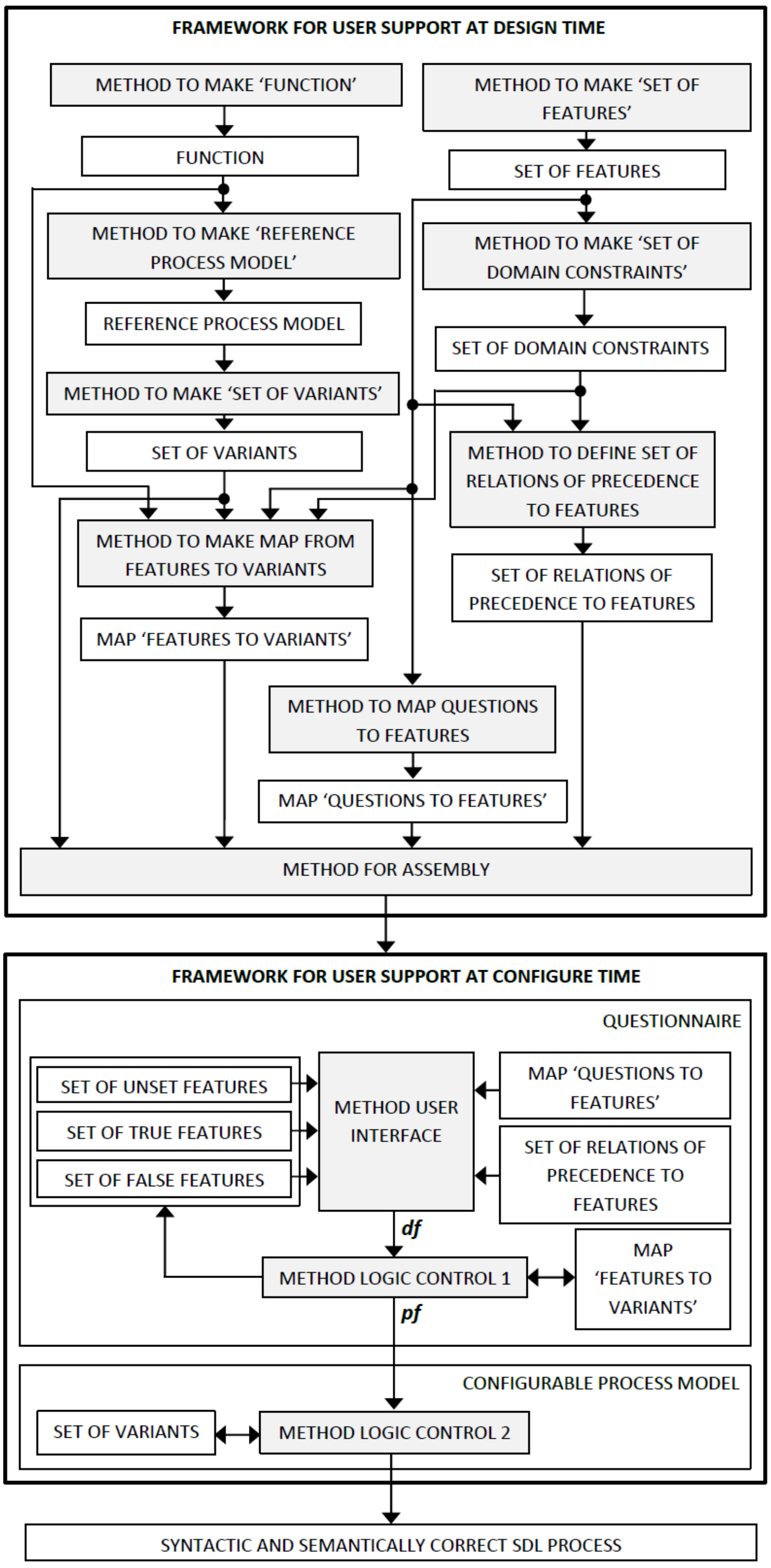

FRAMEWORK FOR USER SUPPORT AT DESIGN TIME
METHOD TO MAKE 'FUNCTION'
METHOD TO MAKE 'SET OF FEATURES'
FUNCTION
SET OF FEATURES
METHOD TO MAKE 'REFERENCE PROCESS MODEL'
METHOD TO MAKE 'SET OF DOMAIN CONSTRAINTS'
REFERENCE PROCESS MODEL
SET OF DOMAIN CONSTRAINTS
METHOD TO MAKE 'SET OF VARIANTS'
SET OF VARIANTS
METHOD TO DEFINE SET OF RELATIONS OF PRECEDENCE TO FEATURES
METHOD TO MAKE MAP FROM FEATURES TO VARIANTS
SET OF RELATIONS OF PRECEDENCE TO FEATURES
MAP 'FEATURES TO VARIANTS'
METHOD TO MAP QUESTIONS TO FEATURES
MAP 'QUESTIONS TO FEATURES'
METHOD FOR ASSEMBLY
FRAMEWORK FOR USER SUPPORT AT CONFIGURE TIME
QUESTIONNAIRE
SET OF UNSET FEATURES
SET OF TRUE FEATURES
SET OF FALSE FEATURES
METHOD USER INTERFACE
MAP 'QUESTIONS TO FEATURES'
SET OF RELATIONS OF PRECEDENCE TO FEATURES
df
METHOD LOGIC CONTROL 1
MAP 'FEATURES TO VARIANTS'
pf
CONFIGURABLE PROCESS MODEL
SET OF VARIANTS
METHOD LOGIC CONTROL 2
SYNTACTIC AND SEMANTICALLY CORRECT SDL PROCESS

Figura 46 – User support framework at design time and User support framework at configure time

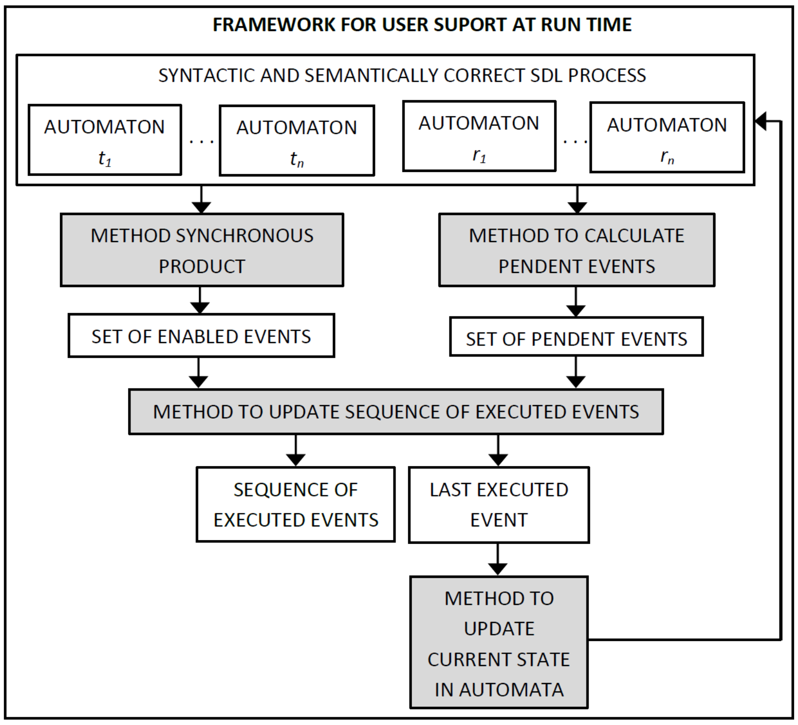

FRAMEWORK FOR USER SUPORT AT RUN TIME
SYNTACTIC AND SEMANTICALLY CORRECT SDL PROCESS
AUTOMATON t_1
AUTOMATON t_n
AUTOMATON r_1
AUTOMATON r_n
METHOD SYNCHRONOUS PRODUCT
METHOD TO CALCULATE PENDENT EVENTS
SET OF ENABLED EVENTS
SET OF PENDENT EVENTS
METHOD TO UPDATE SEQUENCE OF EXECUTED EVENTS
SEQUENCE OF EXECUTED EVENTS
LAST EXECUTED EVENT
METHOD TO UPDATE CURRENT STATE IN AUTOMATA

Figura 47 – User support framework at design time

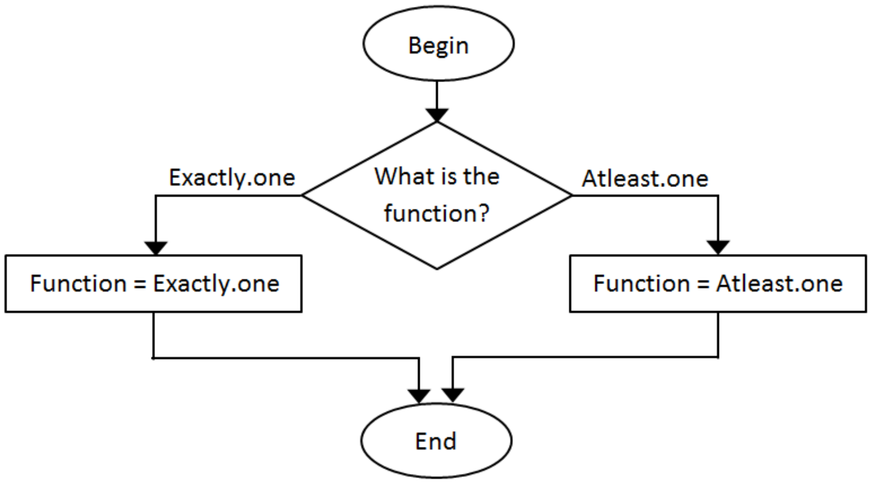


Figura 48 – Method to make *Function*

5.4.1.2 Method to make Set of Features

This method gets no input from other methods. The output from this method is *Set of Features* (SF). SF is defined by the modeler. A feature is a pair (SD, LD), where SD and LD are respectively the short description and the long description for the feature. Each feature is bound to a domain fact. A *domain fact* is a logical variable. So, if the modeler defines $n \geq 1$ features for the process then SF is described by

$$\text{SF} = \{\ (SD_1,\ LD_1) \Leftrightarrow df_1,\ \dots\ ,\ (SD_n,\ LD_n) \Leftrightarrow df_n\ \}$$

where $(SD_i,\ LD_i) \Leftrightarrow df_i$ describes the logic equivalence between *feature i* and *domain fact i*. Thus, *feature i* is true if, and only if, *domain facts i* is true . Figure 49 presents *Method to make Set of Features*. Table 16 describes the steps of *Method to make Set of Features*.

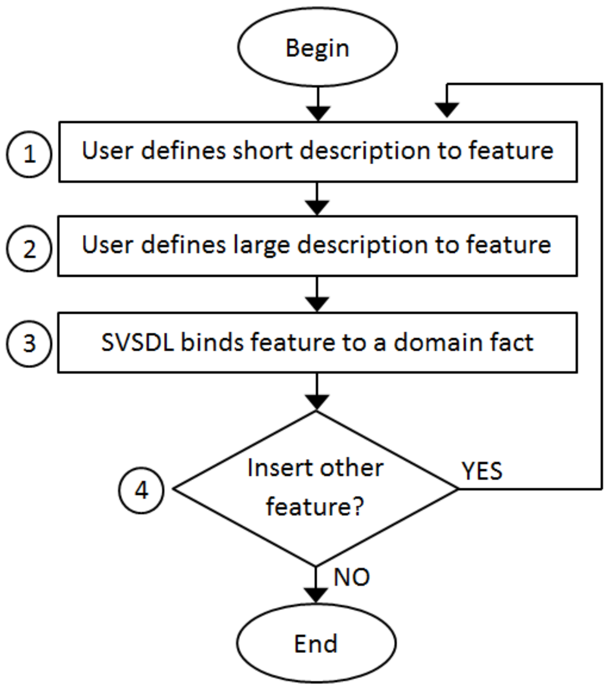

Begin
1
User defines short description to feature
2
User defines large description to feature
3
SVSDL binds feature to a domain fact
4
Insert other feature?
YES
NO
End

Figura 49 – Method to make Set of Features

Tabela 16 – Description of Method to make Set of Features

Step 1 :	User defines the feature in few words. This is the short description of feature.
Step 2 :	User provides longer definition of the feature. This is the long description of the feature.
Step 3 :	SVSDL binds a domain fact (variable) to the structure comprised by the short and long descriptions.
Step 4 :	Users can choose to insert other features or finish the method.

Example 5.4.1. *Let P be a process. Process P produces a given item that can have three different sizes: big, medium, small - with 30mm, 20mm and 10mm, respectively. The colors*

can be black, gray, or white, respectively. And the amount to be produced is 100, 70, and 30, respectively. The short descriptions can be "big", "medium", and "small", and the long description contains the other information. Since there are three features to be inserted, Method to make Set of domain facts makes three domain facts (variables): df_1*,* df_2*, and* df_3*. Thus, the output from Method to make Set of Features is the set SF:*

$$
\begin{aligned}
SF = \{\ &(SD\text{: big, } LD\text{: size} = 30mm,\ color = black,\ amount = 100) \Leftrightarrow df_1,\\
&(SD\text{: medium, } LD\text{: size} = 20mm,\ color = gray,\ amount = 70) \Leftrightarrow df_2,\\
&(SD\text{: small, } LD\text{: size} = 10mm,\ color = white,\ amount = 30) \Leftrightarrow df_3\ \}
\end{aligned}
$$

5.4.1.3 Method to make Set of Domain Constraints

The input to this method is *Set of features*. The output from this method is *Set of domain constraints*. *Set of domain constraints* is a set of constraints defined for features. *Domain constraints* are set by using propositional logic expressions.

Example 5.4.2. *Let SF =* $\{df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}, df_{12}, df_{13}, df_{14}, df_{15}\}$ *be the set of domain facts (features) taken by Method to make Set of constraints for features. For these domain facts, the user defines Set of domain constraints =* $\{DC_1, DC_2, DC_3, DC_4\}$ *such that*

$$DC_1 : df_1 \vee df_2 \vee df_3$$

$$DC_2 : df_3 \veebar df_4 \veebar df_6 \veebar df_7$$

$$DC_3 : \neg df_5 \wedge df_{10} \Leftrightarrow \neg(df_8 \vee df_9)$$

$$DC_4 : \neg df_2 \wedge df_7 \Rightarrow (df_{11} \veebar df_{12})$$

5.4.1.4 Method to make Reference Process Model

The input to this method is *Function*. *Function* can have one of these two values: *Exactly.one* or *Atleast.one*. The output from this method is *Reference Process Model* (RPM). RPM is a process modeled using Simple Declarative Language (SDL) (SCHAIDT; SANTOS, 2017b). SVSDL variants are selected From RPM. A variant from RPM is a syntactically and semantically correct SDL process. This is Definition 5.4.1 presented following.

Definition 5.4.1. *Let A be the RPM in a SVSDL. Let B be a SDL process. B is a variant from A if* $B \subseteq A$ *and B are syntactically and semantically correct.*

Example 5.4.3. *Let A be the RPM in a SVSDL so that T_A = {t_1,t_2} and R_A = {precedence(t_1,t_2), precedence(t_2,t_1)} are the sets of tasks and constraints in A. Let B, C, D and E be four SDL processes such that T_B = {t_1,t_2,t_3} and R_B = {precedence(t_1,t_2), precedence(t_2,t_1)}, T_C = {t_1,t_2} and R_C = {precedence(t_1,t_2), response(t_1,t_2)}, T_D = {t_1,t_2} and R_D = {precedence(t_1,t_2), precedence(t_2,t_1)}, T_E = {t_1,t_2} and R_E = {precedence(t_1,t_2)}. B is a syntactically and semantically correct SDL process but $T_B \not\subseteq T_A$, thus B is not a variant in A. C is a syntactically and semantically correct SDL process but $R_C \not\subseteq R_A$, thus C is not a variant in A. $T_D \subseteq T_A$ and $R_D \subseteq R_A$ but D is a not semantically correct SDL process, thus D is not a variant in A. E is a syntactically and semantically correct SDL process and $T_E \subseteq T_A$ and $R_E \subseteq R_A$, thus E is a variant in A.*

The set of variants defined from RPM is named V_{rpm}. The union of the variants in V_{rpm} is a subset of RPM. The maximum set of variants that can be generated from V_{rpm} is named V_{gen}. V_{gen} is the maximum set of variants from V_{rpm}that, in fact, can be executed. V_{gen} is a function of V_{rpm}. V_{rpm} and V_{gen} are related to each other and this relation is defined by *Function*. V_{rpm}, V_{gen} and their relation, defined by *Function*, is shown in Definitions 5.4.2, 5.4.3 and 5.4.4.

Definition 5.4.2. *The set of variants defined from RPM is named V_{rpm}. The union of variants in V_{rpm} must be a subset of RPM.*

Definition 5.4.3. *V_{gen} is the maximum set of variants that can be generated from V_{rpm}.*

Definition 5.4.4. *V_{gen} and V_{rpm} are related through Function. If Function = Exactly.one, then V_{gen} = Exactly.one(V_{rpm}). If Function = Atleast.one, then V_{gen} = Atleast.one(V_{rpm}). If V_{gen} = Exactly.one(V_{rpm}) then $V_{gen} = V_{rpm}$. If V_{gen} = Atleast.one(V_{rpm}) then V_{gen} = P(V_{rpm}) $\setminus \emptyset$, where P(V_{rpm}) is the power set of V_{rpm}.*

Definitions 5.4.2, 5.4.3 and 5.4.4 are used in Examples 5.4.4 and 5.4.5.

Example 5.4.4. *Let A = (T_A, R_A) be a RPM, such that*

$$T_A = \{t_1, t_2, t_3\},\ R_A = \{atleast(t_1), precedence(t_1,t_2), response(t_2, t_3)\}.$$

From A the user defines two variants: V_1, V_2.
V_1 = (T_1, R_1), such that

$$T_1 = \{t_1, t_2\},\ R_1 = \{atleast(t_1)\},$$

V_2 = (T_2, R_2), such that

$$T_2 = \{t_2, t_3\},\ R_2 = \{response(t_2, t_3)\}.$$

So, V_{rpm} = {V_1, V_2}, and the union of the sets in V_{rpm} is

$$T_U = \{t_1, t_2, t_3\},\ R_U = \{atleast(t_1), response(t_2, t_3)\}.$$

Since (T_U, R_U) $\subseteq$ (T_A, R_A), V_{rpm} is valid.

Example 5.4.5. *Let A be the RPM in a SVSDL. Let B, C and D be the three variants in A. So V_{rpm} = {B, C, D} in A. If Function = Atleast.one then V_{gen} = Atleast.one(V_{rpm}). If V_{gen} = Exactly.one(V_{rpm}) then V_{gen} = {B, C, D}. In other words, if V_{gen} = Exactly.one(V_{rpm}) then there are three variants (B, C, D) that can be selected by the user during configure time. If V_{gen} = Atleast.one(V_{rpm}) then V_{gen} = {B, C, D, $B \cup C$, $B \cup D$, $C \cup D$, $B \cup C \cup D$}. In other words, if V_{gen} = Atleast.one(V_{rpm}) then there are seven variants (B, C, D, B, $B \cup D$, $C \cup D$, $B \cup C \cup D$) that can be selected by the user during configure time.*

Next, we present the Theorems 5.4.1 and 5.4.2. These theorems demonstrate the relations between *Function* and RPM.

Theorem 5.4.1. *Let A be the RPM in a SVSDL. If V_{gen} = Exactly.one(V_{rpm}) then it is not mandatory for A to be semantically correct.*

Proof: If V_{gen} = Exactly.one(V_{rpm}) then two or more variants of V_{rpm} cannot be joined to make a variant in V_{gen}. So, a variant in V_{gen} is always equal to a single variant in V_{rpm}. But, Definition 5.4.1 defines that a variant in V_{rpm} is syntactically and semantically correct. Therefore, every variant in V_{gen} is always syntactically and semantically correct even if A is semantically correct.

Theorem 5.4.2. *Let A be the RPM in a SVSDL. If V_{gen} = Atleast.one(V_{rpm}) then A must be semantically correct.*

Proof: If V_{gen} = Atleast.one(V_{rpm}) then two or more variants from V_{rpm} can be joined to make a variant in V_{gen}. So, a variant in V_{gen} can be equal to a single variant in V_{rpm}, but, a variant in V_{gen} can be also equals to a union of variants from V_{rpm}. Every union of variants from V_{rpm} is contained in V_{rpm}. At the same time, the SDL framework (SCHAIDT; SANTOS, 2017b) guarantees that if an SDL process A is syntactically and semantically correct then every process B, that is syntactically correct and is contained in A, is also semantically correct. Thus, the only way to guarantee that every union of variants from V_{rpm} is semantically correct is Making RPM semantically correct.

After presenting Definitions 5.4.1 to 5.4.4, the sequence of steps for making RPM in SVSDL can be shown. Figure 50 exhibits the sequence of steps for making RPM in SVSDL. Table 17 explains each step of that sequence.

Example 5.4.6. *Let A be the RPM in a SVSDL. Let T_A and R_A be the sets of tasks and constraints of A. Initially, A is empty, thus T_A = $\emptyset$ and R_A = $\emptyset$. Let V_{gen} =*

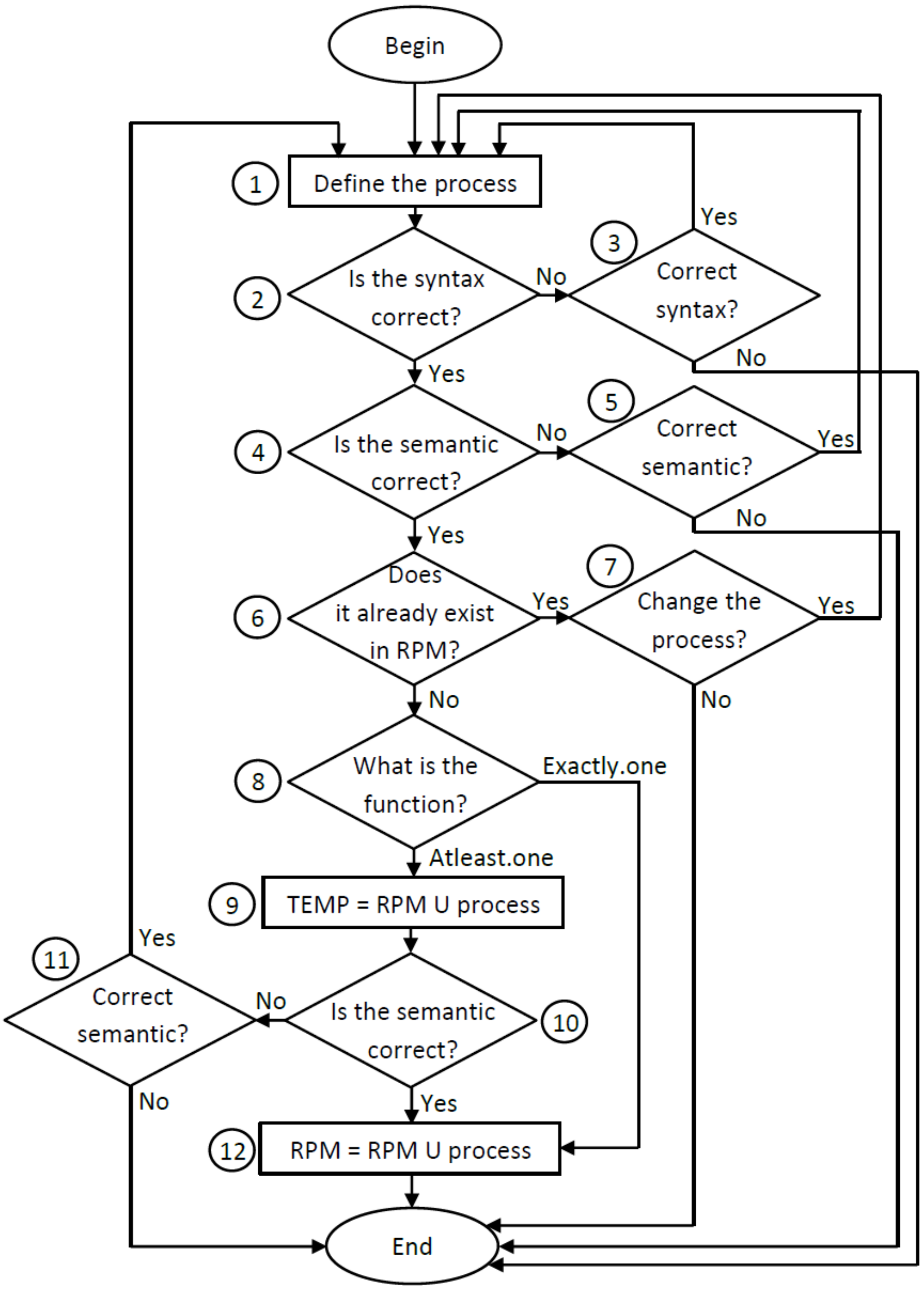


Figura 50 – Method to make a Reference Process Model

Tabela 17 – Description of Method to make a Reference Process Model.

Step 1 :	User defines the sets of tasks and constraints using SDL
Step 2 :	The process defined in step 1 is checked for being syntactically correct, in accordance with Definition 5.4.1. If the check performed in step 2 identifies that the process is not syntactically correct, then the sequence moves to step 3. If the check performed in step 2 identifies that the process is syntactically correct, then the method goes to step 4.
Step 3 :	Users are then asked if they want to correct the syntax of the process. Should users chooses not to correct the syntax of the process, the method comes to an end. If users choose to correct the syntax of the process, then the method returns to step 1.
Step 4 :	The process defined in step 1 is checked for being semantically correct, in accordance with Definition 5.4.1. If the check performed in step 4 identifies that the process is not semantically correct, then the sequence moves to step 5. If the check performed in step 4 identifies that the process is syntactically correct, then the method goes to step 6.
Step 5 :	Users are then asked if they want to correct the semantics of the process. Should users chooses not to correct the semantics of the process, the method comes to an end. If users choose to correct the syntax of the process, then the method returns to step 1.
Step 6 :	The existence of the process defined in step 1 in RPM is checked. Initially, RPM = $\emptyset$. If the check performed in step 6 identifies that the process is equal to an existing process in RPM, the sequence moves to step 7. If the check performed in step 6 identifies that the process is not equal to any existing process in RPM, the sequence moves to step 8.
Step 7 :	Users are asked if they want to change the process. Should users choose not to change the process, then the method comes to an end. If users choose to change the process, the method returns to step 1.
Step 8 :	If $V_{gen} = Atleast.one(V_{rpm})$ or $V_{gen} = Exactly.one(V_{rpm})$ is then checked. If $V_{gen} = Atleast.one(V_{rpm})$, the sequence goes on to step 9. If $V_{gen} = Exactly.one(V_{rpm})$, it moves to step 12.
Step 9 :	A temporary process, called TEMP, is created. TEMP is the union of RPM and the process. After step 9, the sequence carries on to step 10.
Step 10 :	TEMP is checked for being semantically correct. If the check performed in step 10 identifies that TEMP is not semantically correct, the sequence goes to step 11. If the check performed in step 10 identifies that TEMP is semantically correct, then the method proceeds to step 12.
Step 11 :	Users are asked if they want to correct the process. Should users chooses not to correct the process, then the method comes to an end. If users choose to correct the process, the method returns to step 1.
Step 12 :	The process is joined to the RPM. After step 12, the method is concluded.

$Exactly.one(V_{rpm})$ be the function bound to A. As V_{gen} = $Exactly.one(V_{rpm})$, A can be semantically incorrect.

The first process candidate to being in the RPM is P_1 = {T_1, R_1}, where T_1 = {t_1}, R_1 = ∅}. P_1 is syntactically and semantically correct so the checks performed in steps 2 and 4 result in 'YES'. P_1 is not in the RPM yet, so the check carried out in step 6 results in 'No'. Since V_{gen} = $Exactly.one(V_{rpm})$ the check performed in step 8 results in 'Exactly.one'. Thus, the method goes to step 12 where 'RPM = RPM ∪ process'. Making the substitutions results in A = {T_A ∪ T_1, R_A ∪ R_1 }. By continuing, one gets A = {∅ ∪ {t_1}, ∅ ∪ ∅}. And finally, A = {{t_1}, ∅}.

The second process candidate to being in the RPM is P_2 = {T_2, R_2}, where T_2 = {t_2}, R_2 = ∅. P_2 is syntactically and semantically correct so the checks performed in steps 2 and 4 result in 'YES'. P_2 is not in the RPM yet, so the check carried out in step 6 results in 'No'. Since V_{gen} = $Exactly.one(V_{rpm})$ the check performed in step 8 results in 'Exactly.one'. Thus, the method goes to step 12 where 'RPM = RPM ∪ process'. Making the substitutions results in A = {T_A ∪ T_2, R_A ∪ R_2 }. By continuing, one gets A = {{t_1} ∪ {t_2}, ∅ ∪ ∅}. And finally, A = {{t_1, t_2}, ∅}.

The third process candidate to being in the RPM is P_3 = {T_3, R_3}, where T_3 = {t_1, t_2}, R_3 = {precedence(t_1, t_2)}. P_3 is syntactically and semantically correct so the checks performed in steps 2 and 4 result in 'YES'. P_3 is not in the RPM yet, so the check carried out in step 6 results in 'No'. Since V_{gen} = $Exactly.one(V_{rpm})$ the check made in step 8 results in 'Exactly.one'. Thus, the method goes to step 12 where 'RPM = RPM ∪ process'. Making the substitutions results in A = {T_A ∪ T_3, R_A ∪ R_3 }. Continuing, one gets A = {t_1, t_2} ∪ {t_1, t_2}, ∅ ∪ {precedence(t_1, t_2)}}. And finally, A = {{t_1, t_2}, {precedence(t_1, t_2)}}.

The fourth process candidate to being in the RPM is P_4 = {T_4, R_4}, where T_4 = {t_1, t_2}, R_4 = {precedence(t_2, t_1)}. P_4 is syntactically and semantically correct so the checks performed in steps 2 and 4 result in 'YES'. P_4 is not in the RPM yet, so the check carried out in step 6 results in 'No'. Since V_{gen} = $Exactly.one(V_{rpm})$ the check made in step 8 results in 'Exactly.one'. Thus, the method goes to step 12 where 'RPM = RPM ∪ process'. Making the substitutions results in A = {T_A ∪ T_4, R_A ∪ R_4 }. Continuing, one gets A = {t_1, t_2} ∪ {t_1, t_2}, {precedence(t_1, t_2)} ∪ {precedence(t_2, t_1)}}. And finally, A = {{t_1, t_2}, {precedence(t_1, t_2), precedence(t_2, t_1)}}.

So A = {{t_1, t_2}, {precedence(t_1, t_2), precedence(t_2, t_1)}}. Now, the variants in A must be defined. There are 5 possible variants in A: Variant 1 = {{t_1}, ∅}, Variant 2 = {{t_2}, ∅}, Variant 3 = {{t_1, t_2}, ∅}, Variant 4 = {{t_1, t_2}, {precedence(t_1, t_2)}}, and Variant 5 = {{t_1, t_2}, {precedence(t_2, t_1)}}. In this example, all five variants are defined as a variant in A. So V_{rpm} = {Variant 1, Variant 2, Variant 3, Variant 4, Variant 5}. Since V_{gen} = $Exactly.one(V_{rpm})$, we have V_{gen} = V_{rpm}. So V_{gen} = {Variant 1, Variant 2, Variant

3, Variant 4, Variant 5}. This means that users can choose only one variant from V_{rpm} to be performed.

Example 5.4.7. *Let A be the RPM in a SVSDL. Let T_A and R_A be the sets of tasks and constraints of A. Initially, A is empty, thus $T_A = \emptyset$ and $R_A = \emptyset$. Let $V_{gen} = Atleast.one(V_{rpm})$ be the function bound to A. As $V_{gen} = Atleast.one(V_{rpm})$, A must be syntactically and semantically correct.*

The first process candidate to being in A is $P_1 = \{T_1, R_1\}$, where $T_1 = \{t_1, t_2\}$, $R_1 = \{precedence(t_1, t_2)\}$. P_1 is syntactically and semantically correct so the checks carried out in steps 2 and 4 result in 'YES'. P_1 is not in the RPM yet, so the check performed in step 6 results in 'No'. Since $V_{gen} = Atleast.one(V_{rpm})$ the check performed in step 8 results in 'Atleast.one'. Thus, the method goes to step 9 where TEMP = RPM $\cup$ process. Making the substitutions results in $A = \{T_A \cup T_1, R_A \cup R_1\}$, and consequently $A = \{\{t_1, t_2\}, \{precedence(t_1, t_2)\}\}$. As A is semantically correct, the checks performed in step 10 result in 'YES'. The method goes to step 12 where RPM = RPM $\cup$ process. Making the substitutions results in $RPM = \{\emptyset \cup \{t_1, t_2\}, \emptyset \cup \{precedence(t_1, t_2)\}$, and consequently $RPM = \{\{t_1, t_2\}, \{precedence(t_1, t_2)\}$.

The second process candidate to being in A is $P_2 = \{T_2, R_2\}$, where $T_2 = \{t_1, t_2\}$, $R_2 = \{precedence(t_2, t_1)\}$. P_2 is syntactically and semantically correct so the checks carried out in steps 2 and 4 result in 'YES'. P_2 is not in A yet, so the check performed in step 6 results in 'No'. Since $V_{gen} = Atleast.one(V_{rpm})$ the check performed in step 8 results in 'Atleast.one'. Thus, the method goes to step 9 where TEMP = RPM $\cup$ process. Making the substitutions results in $A = \{T_A \cup T_1, R_A \cup R_1\}$, $TEMP = \{\{t_1, t_2\} \cup \{t_1, t_2\}, \{precedence(t_1, t_2)\} \cup \{precedence(t_2, t_1)\}\}$, $TEMP = \{\{t_1, t_2\}, \{precedence(t_1, t_2), precedence(t_2, t_1)\}$. As TEMP is semantically incorrect, the checks performed in step 10 result in 'NO'. Since no changes will be made in P_2, the method comes to an end and P_2 is not inserted in A.

So $A = \{\{t_1, t_2\}, \{precedence(t_1, t_2)\}\}$. Now, the variants in A must be defined. There are 4 possible variants in A: Variant 1 = $\{\{t_1\}, \emptyset\}$, Variant 2 = $\{\{t_2\}, \emptyset\}$, Variant 3 = $\{\{t_1, t_2\}, \emptyset\}$, Variant 4 = $\{\{t_1, t_2\}, \{precedence(t_1, t_2)\}\}$. In this example, all four variants are defined as a variant in A. So V_{rpm} = {Variant 1, Variant 2, Variant 3, Variant 4}. Since $V_{gen} = Atleast.one(V_{rpm})$, we have $V_{gen} = P(V_{rpm}) \setminus \emptyset$, where $P(V_{rpm})$ is the power set of V_{rpm}.

So V_{gen} = {Variant 1, Variant 2, Variant 3, Variant 4, Variant 1 $\cup$ Variant 2, Variant 1 $\cup$ Variant 3, Variant 1 $\cup$ Variant 4, Variant 2 $\cup$ Variant 3, Variant 2 $\cup$ Variant 4, Variant 3 $\cup$ Variant 4, Variant 1 $\cup$ Variant 2 $\cup$ Variant 3, Variant 1 $\cup$ Variant 2 $\cup$ Variant 4, Variant 1 $\cup$ Variant 3 $\cup$ Variant 4, Variant 2 $\cup$ Variant 3 $\cup$ Variant 4, Variant 1 $\cup$ Variant 2 $\cup$ Variant 3 $\cup$ Variant 4}. This means the user can choose more than one variant from V_{rpm} to make the "final variant"to be performed.

5.4.1.5 Method to make Set of Variants (V_{rpm})

The input to this method is *Reference Process Model* (RPM). The output from this method is V_{rpm}. V_{rpm} is the set of variants defined from RPM. V_{rpm} is defined by the modeler.

A variant is a pair (T,R), where T and R are respectively a set of SDL tasks and a set of SDL constraints, and (T,R) is syntactically and semantically correct. Each variant is bound to a process fact. A *process fact* is a logical variable. So, if the modeler defines $n \geq 2$ variants to V_{rpm} then $|V_{rpm}| = n$, and is described by

$V_{rpm} = \{\ (T_1,\ R_1) \Leftrightarrow df_1,\ \ldots\ ,\ (T_n,\ T_n) \Leftrightarrow df_n\ \}$

where $(T_i,\ R_i) \Leftrightarrow df_i$ describes the logic equivalence between *variant i* and *process fact i.* Thus, *variant i* is selected if, and only if, *process fact i* is true. Figure 51 presents *Method to make V_{rpm}*. Table 18 describes the steps of *Method to make V_{rpm}*.

Example 5.4.8. *Let A = (T,R) be the RPM in a SVSDL, so that T = {t_1, t_2, t_3, t_4} and R = {atleast(t_2), atleast(t_3), precedence(t_3,t_1), precedence(t_2,t_4), response(t_3,t_2), response(t_1,t_4)}. Four variants are defined in A, so Vrpm = : ({t_1}, ∅), ({t_2}, ∅), ({t_1, t_2}, ∅), and ({t_1, t_2}, {precedence(t_1, t_2)}). So, V_{rpm} = {({t_1}, ∅), ({t_2}, ∅), ({t_1, t_2}, ∅), ({t_1, t_2}, {precedence(t_1, t_2)})}. Since there are four variants in V_{rpm}, Method to Map Process Facts to Variants makes three process facts (variables): pf_1, pf_2, and pf_3. Thus, the output from Method to Map Process Facts to Variants is the set MPFV presented in the following:*

$$\begin{aligned} MPFV = \{\ & pf_1 \rightarrow (\{t_1,\ t_2,\ t_3,\ t_4\},\ \{atleast(t_2),\ precedence(t_3,t_1),\ precedence(t_2,t_4), \\ & \qquad response(t_3,t_2),\ response(t_1,t_4)\}), \\ & pf_2 \rightarrow (\{t_1,\ t_2,\ t_3,\ t_4\},\ \{atleast(t_3),\ precedence(t_3,t_1),\ response(t_1,t_4)\}), \\ & pf_3 \rightarrow (\{t_1,\ t_2,\ t_3\ \},\ \emptyset)\ \} \end{aligned}$$

5.4.1.6 Method to Map from Features to Variants

The inputs to this method are *Function*, *Set of Variants* (V_{rpm}), *Set of Features*, and *Set of Domain Constraints*. The output of this method is *Map from Features to Variants. Map from Features to Variants* is a set of maps from features in *Set of Features* to variants in *Set of Variants.*

The equivalences among *domain facts* and *process facts* must obey some conditions. These conditions are presented in Definitions 5.4.6 and 5.4.6, and Lemmas 5.4.1 and 5.4.2.

Definition 5.4.5. *Every process fact is logically equivalent to a disjunction of conjunctions of domain facts.*

Tabela 18 – Description of Method to make a Set of variants V_{rpm}.

Step 1 :	Users define the set of tasks and constraints of the SDL process from RPM.
Step 2 :	The SDL process defined in step 1 is checked for syntactic correctness. This check is carried out based on SDL syntax definitions (SCHAIDT; SANTOS, 2017b). If the check performed in step 2 identifies that the process is not syntactically correct, the sequence moves to step 3. If the check performed in step 2 identifies that the process is syntactically correct, the sequence moves to step 4.
Step 3 :	Users are asked if they want to correct the syntax of the process. Should users choose not to correct the syntax of the process, the method comes to an end. If users choose to correct the syntax of the process, the method returns to step 1.
Step 4 :	The process defined in step 1 is checked for semantic correctness. This check is carried out based on SDL semantic definitions (SCHAIDT; SANTOS, 2017b). If the check performed in step 4 identifies that the process is not semantically correct, the sequence moves to step 5. If the check performed in step 4 identifies that the process is semantically correct, the sequence moves to step 6.
Step 5 :	Users are asked if they want to correct the semantics of the process. Should users choose not to correct the semantics of the process, the method comes to an end. If users choose to correct the semantics of the process, the method returns to step 1.
Step 6 :	The process defined in step 1 is checked for already being a variant in RPM. If the check carried out in step 6 identifies that the process is already a variant in RPM, then the sequence goes to step 7. If the check carried out in step 6 identifies the process is not a variant in V_{rpm} yet, then the sequence goes to step 8.
Step 7 :	Users are asked if they want to change the process. Should users choose not to change the process, the method comes to an end. If users choose to change the process, the method returns to step 1.
Step 8 :	A *process fact* is bound to an SDL process in V_{rpm}. This SDL process is now a SDL variant in V_{rpm}. After this step, this method comes to an end.

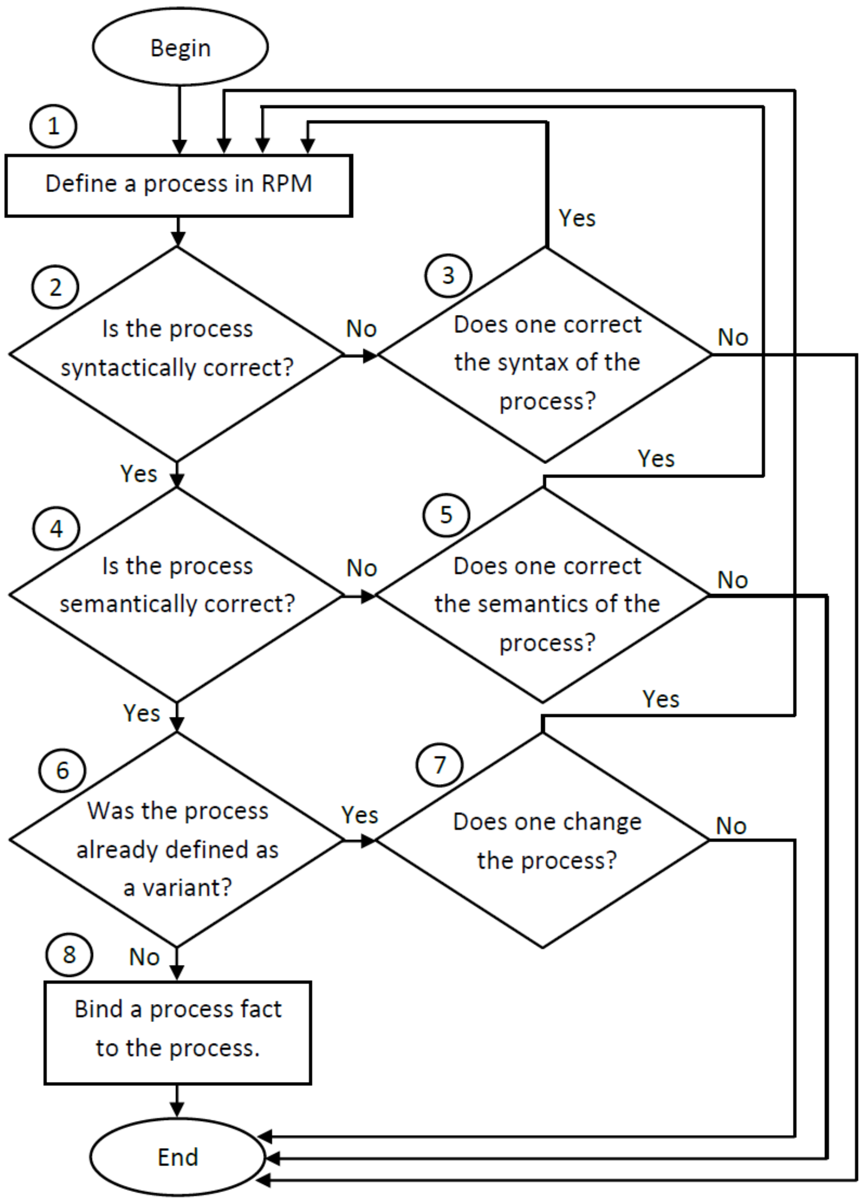

Begin
1
Define a process in RPM
2
Is the process syntactically correct?
No
3
Does one correct the syntax of the process?
Yes
No
Yes
4
Is the process semantically correct?
No
5
Does one correct the semantics of the process?
Yes
No
Yes
6
Was the process already defined as a variant?
Yes
7
Does one change the process?
No
8
No
Bind a process fact to the process.
End

Figura 51 – Method to make a Set of variants (V_{rpm})

Example 5.4.9. *Let A be an RPM. The modeler defines three variants from RPM, so* $|V_rpm| = \{variant\ 1,\ variant\ 2,\ variant3\}$*. This results in making three process facts:* pf_1, pf_2*, and* pf_3*. Each process fact is bound to a variant. The modeler defines six features for the variants in* V_{rpm}*. The features are feature 1, feature 2, feature 3, feature 4, feature 5, feature 6. So, six domain facts are made:* df_1, df_2, df_3, df_4, df_5, df_6*. Each domain fact is bound to a feature. The modeler defines the following mapping from features (domain facts) to variants (process facts):*

$$(df_1 \wedge \neg df_2) \vee (df_3 \wedge df_4) \Leftrightarrow pf_1$$

$$(df_2 \wedge df_5) \vee (df_3 \wedge df_6) \Leftrightarrow pf_2$$

$$df_5 \wedge \neg df_6 \Leftrightarrow pf_3$$

If feature 1 (df_1) is $TRUE$ and feature 2 (df_2) is $FALSE$, or if feature 3 (df_3) and feature 4 (df_4) are $TRUE$, then pf_1 is $TRUE$. If pf_1 is $TRUE$, then variant 1 is selected. This reasoning is valid for all other links between domain facts and process facts.

Definition 5.4.6. *The set of conjunctions of domain facts that are logically equivalent to process facts must be compliant with Set of Domain Constraints.*

Example 5.4.10. *Let SF and DC be the Set of features and Set of domain constrains defined in example 5.4.2. The modeler then defines the following set of equivalences between domain facts and process facts*

$$df_1 \wedge df_2 \wedge df_{10} \Leftrightarrow pf_1$$

$$df_4 \wedge df_6 \wedge \neg df_{15} \Leftrightarrow pf_2$$

$$\neg df_5 \wedge df_{13} \wedge df_{14} \Leftrightarrow pf_3$$

However, this set of equivalences is not valid because conjunction $df_4 \wedge df_6 \wedge \neg df_{15}$ conflicts with $df_3 \veebar df_4 \veebar df_6 \veebar df_7$ (DC_2). Thus, the modeler must be redefine the set of equivalences to avoid this conflict.

Lemma 5.4.1. *If a domain fact is declared in all the conjunctions then there must be a conjunction in which this domain fact is negated, and a conjunction in which it is not negated.*

Demonstração. If a *domain fact* is declared in all the conjunctions and the domain fact is negated in all conjunctions, or is not negated in all conjunctions, then this domain fact is not a logical variable, it is a constant. □

Example 5.4.11. *Let $DF = \{df_1, df_2, df_3, df_4, df_5, df_6, df_7\}$ be the set of domain facts. Let pf_1, pf_2, pf_3 be three process facts. Let*

$$(df_1 \wedge df_2) \vee (df_1 \wedge df_3) \Leftrightarrow pf_1,$$
$$(df_1 \wedge df_4) \vee (df_1 \wedge df_5) \Leftrightarrow pf_2,$$
$$(df_1 \wedge df_6) \vee (df_1 \wedge df_7) \Leftrightarrow pf_3$$

be the equivalences between domain facts and process facts. But df_1 is in all conjunctions and df_1 is not negated in all of them. Thus, df_1 is not a variable, it a is constant. Therefore, df_1 must be excluded from DF. If df_1 is excluded from DF then the equivalences become

$$df_2 \vee df_3 \Leftrightarrow pf_1,$$
$$df_4 \vee df_5 \Leftrightarrow pf_2,$$
$$df_6 \vee df_7 \Leftrightarrow pf_3.$$

Now, let

$(\neg df_1 \wedge \neg df_2) \vee (\neg df_2 \wedge df_3) \Leftrightarrow pf_1,$
$(\neg df_2 \wedge df_4) \vee (\neg df_2 \wedge df_5) \Leftrightarrow pf_2,$
$(\neg df_2 \wedge df_6) \vee (\neg df_2 \wedge df_7) \Leftrightarrow pf_3$

be the equivalences between domain facts and process facts. Then df_2 *is negated in all conjunctions. Thus* df_2 *is not a variable, it is constant. Therefore,* df_2 *must be excluded from* DF*. If* df_2 *is excluded from* DF*, the equivalences become*

$\neg df_1 \vee df_3 \Leftrightarrow pf_1,$
$df_4 \vee df_5 \Leftrightarrow pf_2,$
$df_6 \vee df_7 \Leftrightarrow pf_3.$

Lemma 5.4.2. *If* V_{gen} *= Exclusive.one(*V_{rpm}*) then a conjunction cannot be logically equivalent to more than one process fact.*

Demonstração. If a conjunction is logically equivalent to more than one process fact, it is possible for two variants to be selected to be executed, and this can violate the rules of semantics. □

Example 5.4.12. *Let DF = {*df_1*,* df_2*,* df_3*} be the set of domain facts. Let PF = {*pf_1*,* pf_2*} and* V_{gen} *= Exclusive.one(*V_{rpm}*). The bindings*

$pf_1 \Leftrightarrow df_1 \wedge df_2$
$pf_2 \Leftrightarrow df_1 \wedge df_2 \wedge df_3$

are examples of bindings not allowed because, if $df_1 \wedge df_2 \wedge df_3$ *is TRUE, then* pf_1 *and* pf_2 *are TRUE, and this condition violates function* V_{gen} *= Exclusive.one(*V_{rpm}*).*

After presenting Definitions 5.4.6 and 5.4.6, and Lemmas 5.4.1 and 5.4.2 , *Method to Map from Features to Variants* can be presented. Figure 52 presents *Method to Map from Features to Variants.* Table 19 explains the steps for *Method to Map from Features to Variants.*

5.4.1.7 Method to make a Set of Feature Precedence Relations

The inputs to this method are *Set of Features* and *Set of Domain Constraints.* The output from this method is *Set of Feature Precedence Relations. Set of Feature Precedence Relations* comprises relations that define the order in which features must be set. For defining the relations of precedence among features, this paper proposes function *precedence*(*set A*, *set B*). This function defines that features in *set A* must be set before setting features in *set B.*

Example 5.4.13. *Let precedence({*df_2*,* df_5*,* df_6*}, {*df_1*,* df_3*,* df_4*}) be the precedence relation for features* df_1*,* df_2*,* df_3*,* df_4*,* df_5*,* df_6*. So, at configure time, features* df_2*,* df_5 *and* df_6 *must*

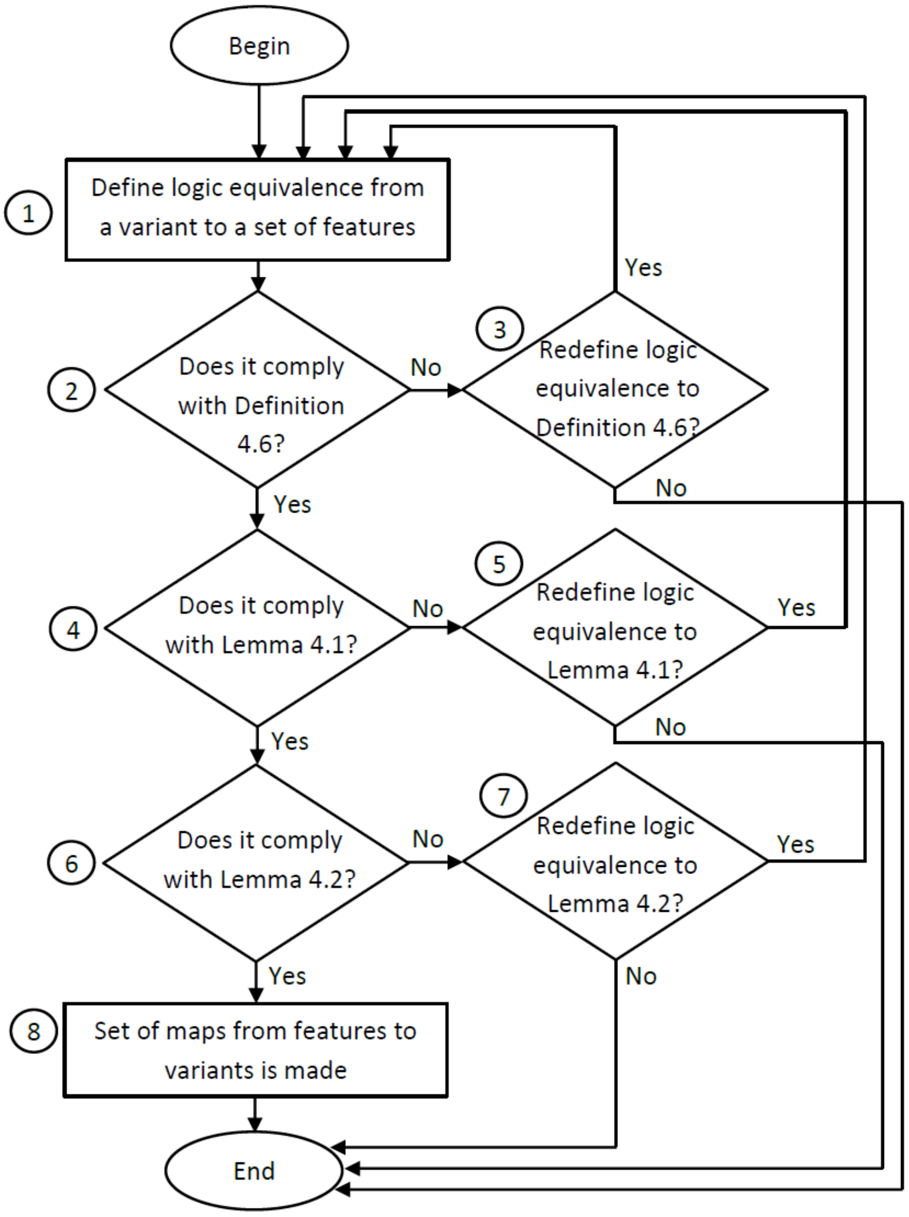

Begin
1
Define logic equivalence from a variant to a set of features
2
Does it comply with Definition 4.6?
No
3
Redefine logic equivalence to Definition 4.6?
Yes
No
Yes
4
Does it comply with Lemma 4.1?
No
5
Redefine logic equivalence to Lemma 4.1?
Yes
No
Yes
6
Does it comply with Lemma 4.2?
No
7
Redefine logic equivalence to Lemma 4.2?
Yes
No
Yes
8
Set of maps from features to variants is made
End

Figura 52 – Method to Map from Features to Variants

be set before features df_1, df_3, df_4 .

5.4.1.8 Method to make a Set of Questions

The input to this method is *Set of Features*. The output from this method is *Map from Questions to Features*. *Set of Questions* is a set of maps from questions to features.

Questions should express contexts in which the features (*domain facts*) are related. The following example 5.4.14 presents an example of mapping among questions and features.

Example 5.4.14. *Let $DF = \{df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8\}$ be the set of domain facts of a process. There are four contexts that relate these eight domain facts: context 1, context*

Tabela 19 – Description of Method to Map from Features to Variants

Step 1 :	Users define the logic equivalence between a variant and a set of features. Users specify conjunctions of features that are logically equivalent to variants
Step 2 :	The equivalence logic is checked for compliance with Set of domain constraints. If the equivalence logic does not comply with Set of domain constraints, the sequence moves to step 3. If the equivalence logic is compliant with Set of domain constraints, the sequence moves to step 4.
Step 3 :	Users are asked if they want to redefine the equivalence logic in order to achieve compliance with Set of domain constraints. Should users choose not to redefine equivalence logic, the method comes to an end. If users choose to redefine equivalence logics, the method returns to step 1.
Step 4 :	Equivalence logic is checked for compliance with Lemma 5.4.1 . If equivalence logic is not compliant with Lemma 5.4.1, the method moves to step 5. If equivalence logic is compliant with Lemma, the sequence goes on to step 6.
Step 5 :	Users are asked if they want to redefine equivalence logic in order to achieve compliance with Lemma 5.4.1. Should users chooses not to redefine equivalence logics, the method comes to an end. If users choose to redefine equivalence logics, the method returns to step 1.
Step 6 :	Equivalence logics are checked for compliance with Lemma 5.4.2. If equivalence logic is not compliant with Lemma 5.4.2, the process goes to step 7. If logic equivalence is compliant with Lemma 5.4.2, the process moves to step 8.
Step 7 :	Users are asked if they want to redefine equivalence logic in order to achieve compliance with Lemma 5.4.2. Should users chooses not to redefine equivalence logics, the method comes to an end. If users choose to redefine equivalence logics, the method returns to step 1.
Step 8 :	A set of maps from features to variants is compiled. Each map is represented by a logic equivalence among features (domain facts) and variants (process facts).

2, context 3 and context 4. Context 1 relates domain facts df_1 and df_2. Context 2 relates domain facts df_3 and df_4. Context 3 relates domain facts df_5 and df_6. Context 4 relates domain facts f_7 and f_8. Thus, the four contexts bring up four questions in such a way that the mapping between questions and domain facts is:

Question 1:
{In context 1, what are the true features and false features?} $\longrightarrow$ *{df_1, df_2}*

Question 2:
{In context 2, what are the true features and false features?} $\longrightarrow$ *{df_3, df_4}*

Question 3:
{In context 3, what are the true features and false features?} $\longrightarrow$ *{df_5, df_6}*

Question 4:
{In context 4, what are the true features and false features?} $\longrightarrow$ *{df_7, df_8}*

In SVSDL, according to Definition 5.4.7, a *domain fact* cannot be bound to more than one questions.

Definition 5.4.7. *A domain fact is bound to only one question.*

Example 5.4.15. *Let $DF = \{df_1, df_2, df_3, df_4, df_5\}$ be the set of domain facts of a process. There are 3 contexts that relate these eight domain facts: context 1, context 2, context 3. Context 1 relates domain facts df_1 and df_2. Context 2 relates domain facts df_2 and df_3. Context 3 relates domain facts df_4 and df_5. Thus, the three contexts bring up three questions so that the mapping between questions and domain facts is:*

Question 1:
{In context 1, what are the true features and false features?} $\longrightarrow$ *{df_1, df_2}*

Question 2:
{In context 2, what are the true features and false features?} $\longrightarrow$ *{df_2, df_3}*

Question 3:
{In context 3, what are the true features and false features?} $\longrightarrow$ *{df_4, df_5}*

But this mapping is not allowed since domain fact df_2 is bound to two questions (Question 1 and Question 2). Thus, the mapping shown in this example must be changed to be permitted.

The sequence to set the *questions* must follow the sequence to set *domain facts.* In other words, the *questions* setting sequence inherits all the precedence relations from *domain facts.*

Example 5.4.16. *Example 5.4.14 provides Question 1 (Q_1), Question 2 (Q_2), Question 3 (Q_3), and Question 4 (Q_4). The assumption is that df_4 and df_5 must be set before df_1 and df_2. So every questions sequence provides for answering Q_3 before Q_1 is permitted. The sequences that do not provide for Q_3 being answered before Q_1 are not permitted.*

Figure 53 presents *Method to make a Set of questions.* Table 20 explains the steps for *Method to make a Set of questions.*

5.4.1.9 Method for Assembly

The inputs for this method are *Set of variants*, *Map from Features to Variants*, *Map from Questions to Features*, *Set of Relations of Precedence of Features.* The output from this

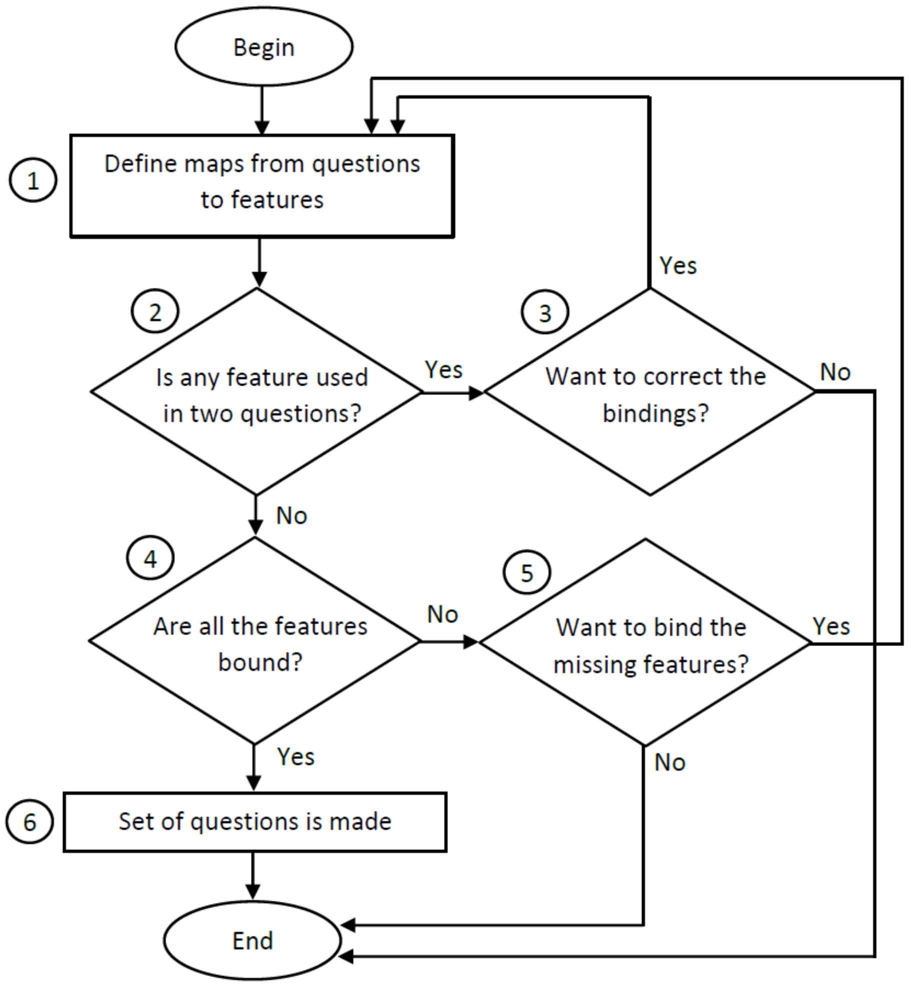

Begin
1
Define maps from questions to features
2
Is any feature used in two questions?
Yes
3
Want to correct the bindings?
Yes
No
No
4
Are all the features bound?
No
5
Want to bind the missing features?
Yes
Yes
No
6
Set of questions is made
End

Figura 53 – Method to to make a Set of questions.

method is *User support framework at configure time.*

Method for Assembly is the only method at design time that does not require user interaction. This method takes all its inputs and automatically make its input. It depends on software routines that run without user participation. However, these software routines are not in the scope of this paper. So, nothing is presented in this section about this method, procedure or routine. Nonetheless, *User support framework at configure time* methods are explained in the following section.

5.4.2 Configure time

At configure time, SVSDL provides users with *User support framework at configure time. User support framework at configure time* produces *syntactically and semantically Correct SDL Processes. User support framework at configure time* is composed by three methods: *Method User Interface*, *Method Logic Control 1*, *Method Logic Control 2*. These methods are described next.

Tabela 20 – Description of Method to make a Set of questions.

Step 1 :	Users define maps from question to features by binding a question to a set of features.
Step 2 :	The maps are checked for two different questions bound to same feature. If two different questions are bound to the same feature, the process moves to step 3. If no two questions are bound to same feature, the process moves to step 4.
Step 3 :	Users are asked if they want to correct the bindings in order to avoid two questions from being bound to the same feature. Should users choose not to correct the bindings, the method comes to an end. If users choose to correct the bindings, the method returns to step 1.
Step 4 :	Features are checked for being bound to one question. If any feature is not bound to a question, the method moves to step 5. If every feature is bound to a question, the method moves forward to step 6.
Step 5 :	Users are asked if they want to bind the missing features. Should users choose not to bind the missing features, the method comes to an end. If users choose to bind the missing features, the method returns to step 1.
Step 6 :	A set of maps from features to variants is compiled. Each map is represented by a link between a given question and a set of features.

5.4.2.1 Method User Interface

This method provides a user interface for selection of *domain facts* (features). After users select a *domain fact* (feature), the *domain fact* is sent to *Method Logic Control 1*. Figure 54 presents *Method User Interface*. Table 21 explains the steps from *Method User Interface*.

5.4.2.2 Method Logic Control 1

This method receives a *domain fact* from *Method User Interface* and checks if a *process fact* (variant) has been selected. If a *process fact* (variant) has been selected, the *process fact* is sent to *Method Logic Control 2*. Figure 55 shows *Method Logic Control 1*. Table 22 explains the steps for *Method Logic Control 1*.

5.4.2.3 Method Logic Control 2

This method receives a *process fact* from *Method Logic Control 1* and selects the variant (SDL process) that is bound to this *process fact*. Figure 56 presents the steps in *Method Logic Control 2*. Table 23 describes the steps of *Method Logic Control 2*.

5.4.3 Run time

At run time, SVSDL provides users with a *User support framework at run time*. *User support framework at run time* receives the variant (an SDL process) from *User support*

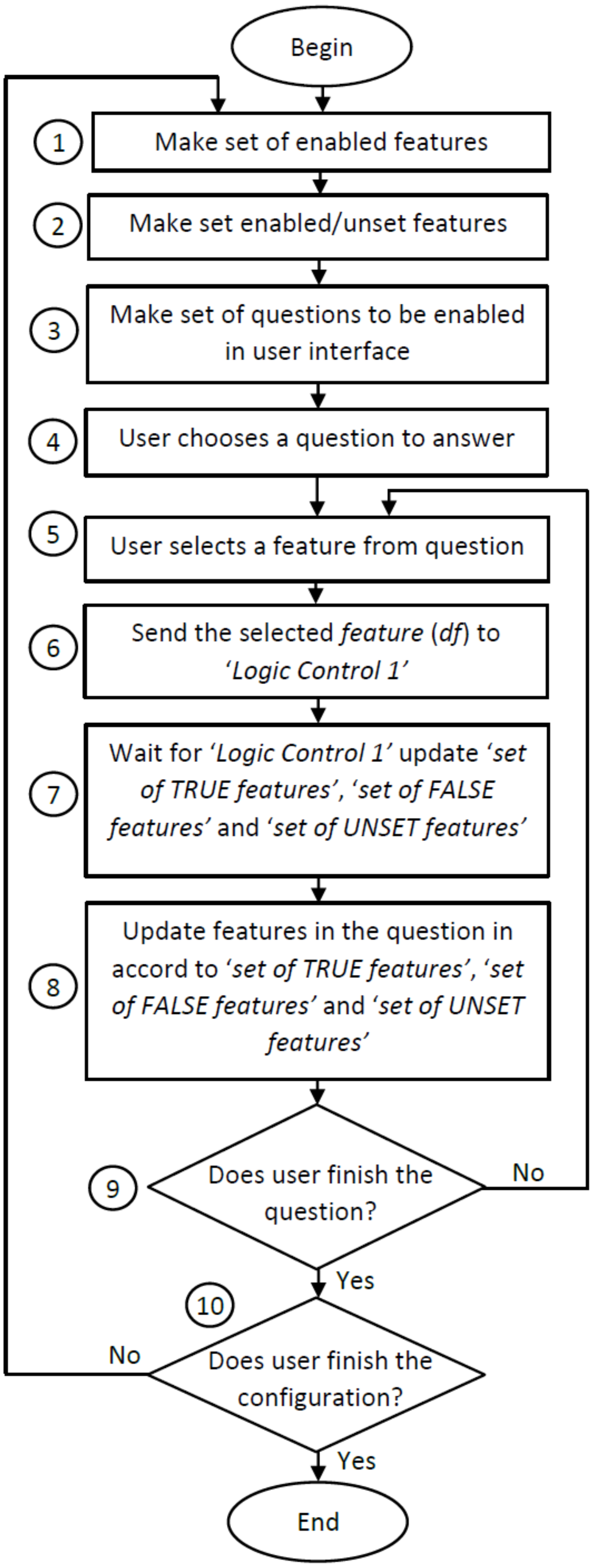

Begin
1
Make set of enabled features
2
Make set enabled/unset features
3
Make set of questions to be enabled in user interface
4
User chooses a question to answer
5
User selects a feature from question
6
Send the selected *feature* (*df*) to *'Logic Control 1'*
7
Wait for *'Logic Control 1'* update *'set of TRUE features'*, *'set of FALSE features'* and *'set of UNSET features'*
8
Update features in the question in accord to *'set of TRUE features'*, *'set of FALSE features'* and *'set of UNSET features'*
9
Does user finish the question?
No
Yes
10
Does user finish the configuration?
No
Yes
End

Figura 54 – Method user interface

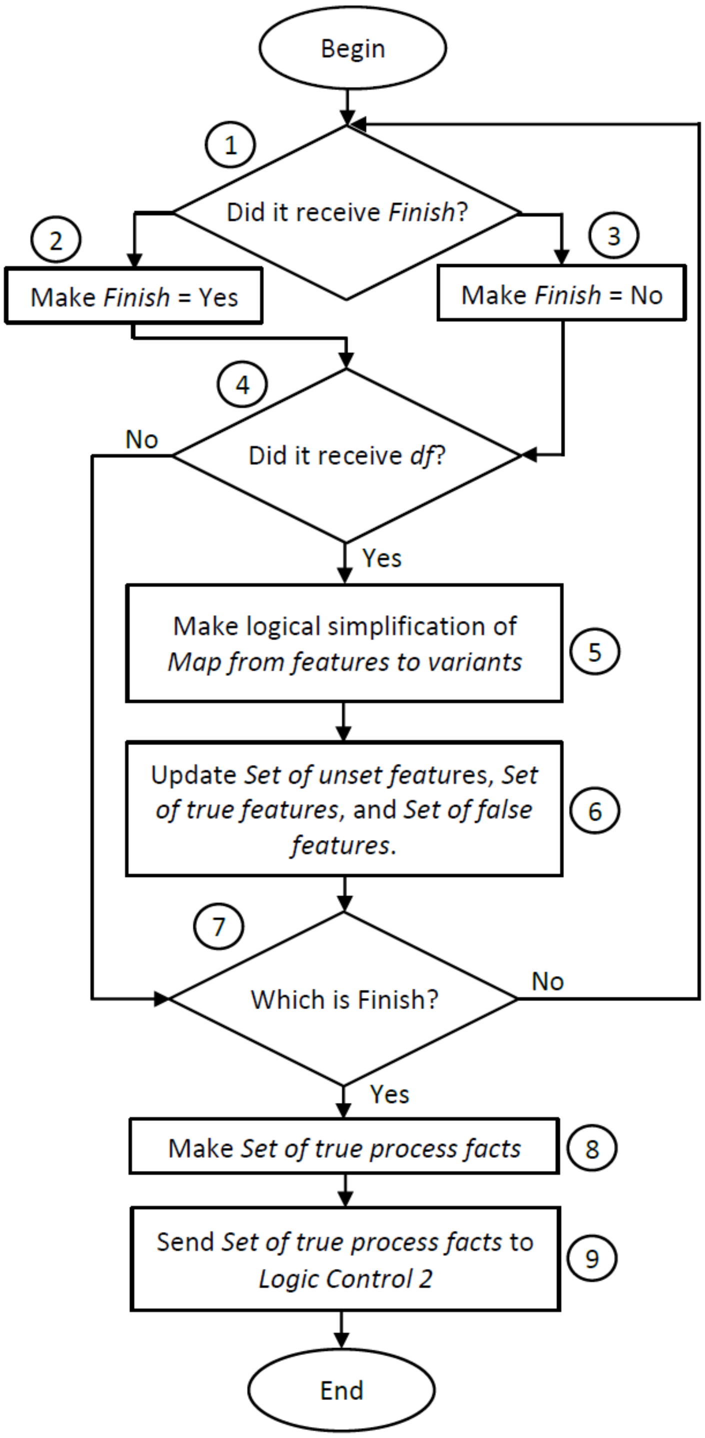

Begin
1
Did it receive *Finish*?
2
Make *Finish* = Yes
3
Make *Finish* = No
4
No
Did it receive *df*?
Yes
Make logical simplification of *Map from features to variants*
5
Update *Set of unset features*, *Set of true features*, and *Set of false features*.
6
7
Which is Finish?
No
Yes
Make *Set of true process facts*
8
Send *Set of true process facts* to *Logic Control 2*
9
End

Figura 55 – Method Logic Control 1

Tabela 21 – Description of Method User Interface

Step 1 :	The set of feature precedence relations is checked. This is done to identify the features that can be set in line with the precedence relations defined by users at design time. The set of features that can be set in line with the precedence relations is called Set of enabled features.
Step 2 :	This set establishes the intersection between *Set of enabled features* and *Set of unset features*resulting in *Set enabled/unset features*.
Step 3 :	The questions bound to the features in *Set enabled/unset features* are identified. These questions and their features are displayed in the user interface.
Step 4 :	Users choose a question to be answered from user interface.
Step 5 :	Users select the *feature i* from the question.
Step 6 :	The interface sends df_i to *Method Logic Control 1*.
Step 7 :	Then, it waits for *Method Logic Control 1* to update *set of true features*, *set of false features* and *set of unset features*.
Step 8 :	After *Method Logic Control 1* has updated the sets specified in step 7, it updates the values of features in the question being answered by the user. This is necessary because when users select a feature, other features may have their values changed.
Step 9 :	Users can choose whether to select other features from this question or to finish it. Should users choose to select other features from this question, the process cycles back to step 5. If users choose to finish the question, the methd moves on to step 10.
Step 10 :	Users can choose whether to select another question to answer or finish the configuration procedure. Should users choose to select other questions to answer, the procedures cycles back to step 1. If users choose to finish the configuration procedure, a Finish message is sent to *Method Logic Control 1*, and the operation is ended.

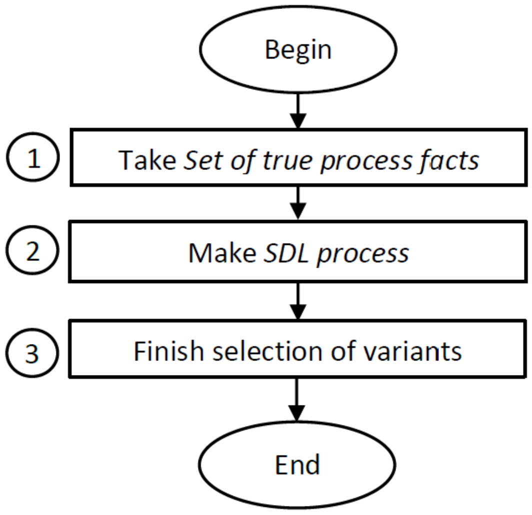

Begin
1
Take *Set of true process facts*
2
Make *SDL process*
3
Finish selection of variants
End

Figura 56 – Method logic control 2

Tabela 22 – Description of Method Logic control 1.

Step 1 :	The method checks whether *Finish* has been received from Method User Interface. If *Finish* is received, then the process moves to step 2. If *Finish* is not received, it moves forward to step 3.
Step 2 :	Internal variable *Finish* = *Yes* is prepared.
Step 3 :	Internal variable *Finish* = *No* is prepared.
Step 4 :	The system then checks for reception of a variable *domain fact* set as *TRUE* or *FALSE*. If a variable *domain fact* has been received,the process moves on to step 5. If no variable *domain fact* has been receives, then it moves to step 7.
Step 5 :	The method performs logic simplification for *Map from features to variants*. Since *Map from features to variants* is in fact comprised of logical equivalences, this step simplifies and reduces these logical equivalences. This identifies the domain facts that are TRUE and FALSE as consequence of the user-defined domain facts. Another consequence of this step is establishing the domain facts that are *TRUE*.
Step 6 :	As a consequence of logic simplification in Step 5, new *domain facts* are set. So, *Set of true features*, *Set of false features* and *Set of unset features* must be updated.
Step 7 :	Internal variable *Finish* = *Yes* or *No* is checked. If internal variable *Finish* = *Yes*, the process goes on to Step 8. If internal variable *Finish* = *No*, it cycles back to Step 1.
Step 8 :	A *Set of true process facts* is compiled. *Set of true process facts* is comprised of all process facts that are *TRUE* in *Map from features to variants*.
Step 9 :	The *Set of true process facts* is then sent to *Method Logic Control 2*.

Tabela 23 – Description of Method Logic control 2.

Step 1 :	The method takes *Set of true process facts* from *Logic Control 1*.
Step 2 :	It the unites all the variants in *Set of Variants* that are bound to process facts in *Set of true process facts*. This union is a *syntactically and semantically Correct SDL Process* that is provided to *User support framework at run time*.
Step 3 :	The selection of variants is ended.

framework at configure time and provides users with methods to run that variant (SDL process). *User support framework at run time* is in fact *SDL framework at run time* (SCHAIDT; SANTOS, 2017b). Since *SDL framework at run time* was described in a previous paper, and *SDL framework at run time* and *User support framework at run time* are the same, no other description or explanation about *User support framework at run time* is required here.

5.4.4 Two examples

This section presents two *User support framework at design time*, *User support framework at configure time* and *User support framework at run time* operation examples. The first example is given by using $Function = Exactly.one$ and the second example is given by using $Function = Atleast.one$. The operations at design and configure time are described using steps. These steps are used for the purpose of making the sequence of operations clearer.

5.4.4.1 Example 1: *Function = Exactly.one*

This section presents an operational example of the three frameworks deployed when users define $Function = Exactly.one$. In this case, users can only select one variant.

5.4.4.1.1 At design time

Step 1: Using *Method to define Function*, users define $Function = Exactly.one$.

Step 2: Using *Method to make Reference Process Model*, users define $RPM = (T,R)$, where
$T = \{t_1, t_2, t_3, t_4, t_5\}$,
$R = \{atleast(t_1), atleast(t_5), precedence(t_1,t_2), precedence(t_2,t_1), response(t_3,t_4)\}$.

Step 3: Using *Method to make Set of Variants*, users define four variants:

$pf_1 : (\{t_1, t_2, t_5\}, \{atleast(t_1), precedence(t_1,t_2)\})$

$pf_2 : (\{t_1, t_2, t_3\}, \emptyset)$

$pf_3 : (\{t_3, t_4, t_5\}, \{atleast(t_5), response(t_3,t_4)\})$

$pf_4 : (\{t_1, t_2, t_3, t_4, t_5\}, \{atleast(t_5), precedence(t_2,t_1)\})$.

Step 4: By *Method to make Set of Features*, users define four features:

df_1 : *feature 1*

df_2 : *feature 2*

df_3 : *feature 3*

df_4 : *feature 4*.

Step 5: Using *Method to make Set of Domain Constraints*, users define *domain constraint* DC_1:

$DC_1 : df_1 \vee df_2 \Rightarrow df_3 \vee df_4$.

By simplifying the logic in DC_1, it can be also described as:

$DC_1 : (\neg df_1 \wedge \neg df_2) \vee df_3 \vee df_4$.

Step 6: Since DC_1 defines that if df_1 or df_2 are set as TRUE, df_3 or df_4 must be set as TRUE, users define that df_1 and df_2 must be set before df_3 and df_4 are set. Thus, by *Method to make Set of Feature Precedence Relations*, users define:

Set of Feature Precedence Relations $= \{precedence(\{df_1, df_2\}, \{df_3, df_4\})\}$.

Step 7: Using *Method to Map from Features to Variants*, user defines *Map from Features to Variants*:

$$\begin{aligned} \textit{Map from Features to Variants} = \{ & df_1 \wedge df_2 \wedge df_3 \wedge \neg df_4 \Leftrightarrow pf_1, \\ & df_1 \wedge df_2 \wedge \neg df_3 \wedge df_4 \Leftrightarrow pf_2, \\ & \neg df_1 \wedge df_2 \wedge df_3 \wedge df_4 \Leftrightarrow pf_3, \\ & \neg df_1 \wedge \neg df_2 \wedge \neg df_3 \wedge df_4 \Leftrightarrow pf_4 \}. \end{aligned}$$

Step 8: Using *Method to Map from Questions to Features*, users define that *Question 1* is bound to features df_1 and df_2, and *Question 2* is bound to features df_3 and df_4. Thus,

$$\begin{aligned} \textit{Map from Questions to Features} = \{ & (\textit{Question 1}\ (Q_1) : \{df_1, df_2\}) \ , \\ & (\textit{Question 2}\ (Q_2) : \{df_3, df_4\}) \ \} \end{aligned}$$

Step 9: From the previously defined inputs, *Method for Assembly* generates the *User support framework at configure time. User support framework at configure time* is used at configure time.

5.4.4.1.2 At configure time

Step 1: *Method User Interface* checks *Set of Feature Precedence Relations*. *Set of Feature Precedence Relations* provides only one relation of precedence: *precedence*($\{df_1, df_2\}$, $\{df_3, df_4\}$). Applying this, *Method User Interface* defines *Set of Enabled Features* = $\{df_1, df_2\}$.

Step 2: *Method User Interface* creates the intersection between *Set of Enabled Features* and *Set of Unset Features*, which is the *Set of Enabled/Unset Features*. *Set of Unset Features* = $\{df_1, df_2, df_3, df_4\}$ because no feature has been set yet. So,

Set of Enabled/Unset Features = $\{df_1, df_2\}$.

Step 3: *Method User Interface* checks which questions are bound to features in *Set of Enabled/Unset Features*. There is only one question bound to the features in *Set of Enabled/Unset Features*: *Question 1* (Q_1). So, Q_1 is displayed to users:

Question 1: df_1: ( ) *TRUE* ( ) *FALSE*

df_2: ( ) *TRUE* ( ) *FALSE*

Step 4: Using *Method User Interface*, users select whether df_1 is *TRUE*:

Question 1: df_1: (X) *TRUE* ( ) *FALSE*

df_2: ( ) *TRUE* ( ) *FALSE*

Step 5: *Control Logic 1* receives df_1 =*TRUE* and performs simplification of *Map from Features to Variants*. Since df_1 =*TRUE*, pf_3 and pf_4 are automatically *FALSE*. So, *Map from Features to Variants* becomes

$$\begin{aligned} \textit{Map from Features to Variants} = \{\ & df_2 \wedge df_3 \wedge \neg df_4 \Leftrightarrow pf_1, \\ & df_2 \wedge \neg df_3 \wedge df_4 \Leftrightarrow pf_2, \\ & FALSE \Leftrightarrow pf_3, \\ & FALSE \Leftrightarrow pf_4\ \}. \end{aligned}$$

But simplification must ensure that at least one variant is selected, so df_2 must be TRUE because it is in all conjunctions. Thus,

$$\begin{aligned} \textit{Map from Features to Variants} = \{\ & df_3 \wedge \neg df_4 \Leftrightarrow pf_1, \\ & \neg df_3 \wedge df_4 \Leftrightarrow pf_2, \\ & FALSE \Leftrightarrow pf_3, \\ & FALSE \Leftrightarrow pf_4\ \}. \end{aligned}$$

Step 6: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features.* So,

Set of Unset Features = { df_3, df_4 }

Set of True Features = { df_1, df_2 }

Set of False Features = $\emptyset$

Step 7: *Method User Interface* updates the value of features in Q_1. Thus, df_1 and df_2 are shown *TRUE* in Q_1:

Question 1: df_1: (X) *TRUE* ( ) *FALSE*

df_2: (X) *TRUE* ( ) *FALSE*

Step 8: Since the user cannot set any other feature in Q_1, he/she chooses to close Q_1.

Step 9: *Method User Interface* checks *Set of Feature Precedence Relations. Set of Feature Precedence Relations* = {*precedence*({df_1, df_2}, {df_3, df_4})}. Since df_1 and df_2 are set, *Method User Interface* defines *Set of Enabled Features* = {df_3, df_4}.

Step 10: *Method User Interface* creates the intersection between *Set of Enabled Features* and *Set of Unset Features.* This is *Set of Enabled/Unset Features. Set of Unset Features* = {df_3, df_4}. So, *Set of Enabled/Unset Features* = {df_3, df_4}.

Step 11: *Method User Interface* checks which questions bound to features in *Set of Enabled/Unset Features.* There is only one question bound to features in *Set of Enabled/Unset Features*: *Question 2* (Q_2). So, Q_2 is displayed to users:

Question 2: df_3: ( ) *TRUE* ( ) *FALSE*

df_4: ( ) *TRUE* ( ) *FALSE*

Step 12: Users select df_3 is *TRUE*:

Question 2: df_3: (X) *TRUE* ( ) *FALSE*

df_4: ( ) *TRUE* ( ) *FALSE*

Step 13: *Control Logic 1* receives df_3 =*TRUE* and performs simplification of *Map from Features to Variants.* Since df_3 =*TRUE*, pf_2 is *FALSE.* So,

$$\begin{aligned} \textit{Map from Features to Variants} = \{\ & \neg df_4 \Leftrightarrow pf_1, \\ & FALSE \Leftrightarrow pf_2, \\ & FALSE \Leftrightarrow pf_3, \\ & FALSE \Leftrightarrow pf_4\ \}. \end{aligned}$$

But simplification must ensure that at least one variant is selected, so df_4 must be FALSE because it is in all conjunctions. Thus, *Map from Features to Variants* becomes

$$\begin{aligned} \textit{Map from Features to Variants} = \{\ & TRUE \Leftrightarrow pf_1, \\ & FALSE \Leftrightarrow pf_2, \\ & FALSE \Leftrightarrow pf_3, \\ & FALSE \Leftrightarrow pf_4\ \}. \end{aligned}$$

Step 14: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features $= \emptyset$

Set of True Features $= \{\ df_1,\ df_2,\ df_3\ \}$

Set of False Features $= \{\ df_4\ \}$

Step 15: *Method User Interface* updates the value of features in Q_2:

Question 2: df_3: (X)*TRUE* ( )*FALSE*

df_4: ( )*TRUE* (X)*FALSE*

Step 16: Since users cannot set any other feature in Q_2, they choose to close Q_1.

Step 17: Users choose to finish the selection of variants.

Step 18: *Method Control Logic 1* makes *Set of true process facts* $= \{pf_1\}$ and sends it to *Method Control Logic 2*.

Step 19: *Method Control Logic 2* sets as *TRUE* the variants in *Set of Variants* that are bound to *process facts* in *Set of true process facts*. So,

$$\begin{aligned} \textit{Set of Variants} = \{\ & TRUE : (\{t_1,\ t_2,\ t_5\},\ \{atleast(t_1),\ precedence(t_1,t_2)\}), \\ & pf_2 : (\{t_1,\ t_2,\ t_3\},\ \emptyset), \end{aligned}$$

pf_3 : ({t_3, t_4, t_5}, {*atleast*(t_5), *response*(t_3,t_4)}),

pf_4 : ({t_1, t_2, t_3, t_4, t_5}, {*atleast*(t_5), *precedence*(t_2,t_1)}) }

Step 20: *Method Control Logic 2* joins together all variants in *Set of Variants* that are set as *TRUE*. This is a *syntactically and semantically Correct SDL Process* to be provided to *User support framework at run time.* So,

SDL Process = ({t_1, t_2, t_5}, {*atleast*(t_1), *precedence*(t_1,t_2)})

Step 21: *Method Control Logic 2* finishes the selection of variants.

5.4.4.1.3 At run time

At run time, *User support framework at run time* receives *syntactically and semantically Correct SDL Process* provided by *User support framework at configure time*. As mentioned previously, *User support framework at run time* is the same as *SDL framework at run time*. Since the operation of *SDL framework at run time* has already been described and an exemple provided (SCHAIDT; SANTOS, 2017b), no demonstration of its operation is made for the purpose of this example.

5.4.4.2 Example 2: *Function* = *Atleast.one*

This section presents an operational example of the three frameworks when users define *Function* = *Atleast.one*. In this case, users can select more than one variant.

5.4.4.2.1 At design time

Step 1: Using *Method to define Function*, users define *Function* = *Atleast.one*.

Step 2: Using *Method to make Reference Process Model*, users define $RPM = (T,R)$, where

$T = \{t_1, t_2, t_3, t_4, t_5\}$,
R = {*atleast*(t_1), *atleast*(t_5), *precedence*(t_1,t_2), *precedence*(t_2,t_1), *response*(t_3,t_4)}.

Step 3: Using *Method to make Set of Variants*, users define four variants:

pf_1 : ({t_1, t_2, t_5}, {*atleast*(t_1), *precedence*(t_1,t_2)})

pf_2 : ({t_1, t_2, t_3}, $\emptyset$)

pf_3 : ({t_3, t_4, t_5}, {*atleast*(t_5), *response*(t_3,t_4)})

pf_4 : ({t_1, t_2, t_3, t_4, t_5}, {*atleast*(t_5), *precedence*(t_1,t_2)}).

pf_5 : ({t_1, t_2, t_3, t_4}, {*atleast*(t_1), *precedence*(t_1,t_2)})

Step 4: Using *Method to make Set of Features*, users define four features:

df$_1$: *feature 1*

df$_2$: *feature 2*

df$_3$: *feature 3*

df$_4$: *feature 4*.

Step 5: Using *Method to make Set of Domain Constraints*, users define *domain constraint* DC_1:

DC_1 : $df_1 \lor df_2 \Rightarrow df_3 \lor df_4$.

By applying logic simplification oo DC_1, this can be also described as:

DC_1 : $(\neg df_1 \land \neg df_2) \lor df_3 \lor df_4$.

Step 6: Since DC_1 defines that if df_1 or df_2 are set as TRUE, df_3 or df_4 must be set as TRUE, users define that df_1 and df_2 must be set before df_3 and df_4 are set. Thus, using *Method to make Set of Feature Precedence Relations*, users define:

Set of Feature Precedence Relations = {*precedence*({df_1, df_2}, {df_3, df_4})}.

Step 7: By *Method to Map from Features to Variants*, users define *Map from Features to Variants*:

Map from Features to Variants = { $df_1 \land df_2 \Leftrightarrow pf_1$,

$df_1 \land df_3 \Leftrightarrow pf_2$,

$df_1 \land df_4 \Leftrightarrow pf_3$,

$df_2 \Leftrightarrow pf_4$,

$df_2 \land df_4 \Leftrightarrow pf_5$ }.

Step 8: By *Method to Map from Questions to Features*, users define that *Question 1* is bound to features df_1 and df_2, and *Question 2* is bound to features df_3 and df_4. Thus,

Map from Questions to Features = { (*Question 1* (Q_1) : {df_1, df_2}) ,

(*Question 2* (Q_2) : {df_3, df_4}) }

Step 9: From previously defined inputs, *Method for Assembly* makes *User support framework at configure time. User support framework at configure time* is used at configure time.

5.4.4.2.2 At configure time

Step 1: *Method User Interface* checks *Set of Feature Precedence Relations*. *Set of Feature Precedence Relations* provides only one relation of precedence: *precedence*($\{df_1, df_2\}$, $\{df_3, df_4\}$). Thus, *Method User Interface* defines *Set of Enabled Features* $= \{df_1, df_2\}$.

Step 2: *Method User Interface* performs the intersection between *Set of Enabled Features* and *Set of Unset Features*. This is *Set of Enabled/Unset Features*. *Set of Unset Features* $= \{df_1, df_2, df_3, df_4\}$ since no feature has been set yet. So, *Set of Enabled/Unset Features* $= \{df_1, df_2\}$.

Step 3: *Method User Interface* checks the questions bound to features in *Set of Enabled/Unset Features*. There is only one question bound to features in *Set of Enabled/Unset Features*: *Question 1* (Q_1). So, Q_1 is displayed to users:

Question 1: df_1: ( ) *TRUE* ( ) *FALSE*

df_2: ( ) *TRUE* ( ) *FALSE*

Step 4: Using *Method User Interface*, users select df_2 as *TRUE*:

Question 1: df_1: ( ) *TRUE* ( ) *FALSE*

df_2: (X) *TRUE* ( ) *FALSE*

Step 5: *Control Logic 1* receives $df_2 = TRUE$ and performs simplification of *Map from Features to Variants*. Since $df_2 = TRUE$, *Map from Features to Variants* becomes

$$\textit{Map from Features to Variants} = \{\, df_1 \Leftrightarrow pf_1,\; df_1 \wedge df_3 \Leftrightarrow pf_2,\; df_1 \wedge df_4 \Leftrightarrow pf_3,\; TRUE \Leftrightarrow pf_4,\; df_4 \Leftrightarrow pf_5 \,\}.$$

Step 6: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features $= \{ df_1, df_3, df_4 \}$

Set of True Features $= \{ df_2 \}$

Set of False Features $= \emptyset$

Step 7: *Method User Interface* updates the value of features in Q_1:

Question 1: df_1: ( ) *TRUE* ( ) *FALSE*

df_2: (X) *TRUE* ( ) *FALSE*

Step 8: Using *Method User Interface*, users set df_1 as *TRUE*:

Question 1: df_1: (X) *TRUE* ( ) *FALSE*

df_2: (X) *TRUE* ( ) *FALSE*

Step 9: *Control Logic 1* receives $df_1 = TRUE$ and performs simplification of *Map from Features to Variants*:

Map from Features to Variants = $\{ TRUE \Leftrightarrow pf_1,$

$df_3 \Leftrightarrow pf_2,$

$df_4 \Leftrightarrow pf_3,$

$TRUE \Leftrightarrow pf_4,$

$df_4 \Leftrightarrow pf_5 \}.$

Step 10: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features = $\{df_3, df_4\}$

Set of True Features = $\{df_1, df_2\}$

Set of False Features = $\emptyset$

Step 11: *Method User Interface* updates the value of features in Q_1:

Question 1: df_1: (X) *TRUE* ( ) *FALSE*

df_2: (X) *TRUE* ( ) *FALSE*

Step 12: Since users cannot set any other feature in Q_1, users choose to close Q_1.

Step 13: *Method User Interface* checks *Set of Feature Precedence Relations*. *Set of Feature Precedence Relations* = $\{precedence(\{df_1, df_2\}, \{df_3, df_4\})\}$. Since df_1 and df_2 are set, *Method User Interface* defines *Set of Enabled Features* = $\{df_3, df_4\}$.

Step 14: *Method User Interface* performs the intersection between *Set of Enabled Features* and *Set of Unset Features*. This is *Set of Enabled/Unset Features*. *Set of Unset Features* = $\{df_3, df_4\}$. So, *Set of Enabled/Unset Features* = $\{df_3, df_4\}$.

Step 15: *Method User Interface* checks the questions bound to features in *Set of Enabled/Unset Features*. There is only one question bound to features in *Set of Enabled/Unset Features*: *Question 2* (Q_2). So, Q_2 is presented to the user:

Question 2: df_3: ( )*TRUE* ( )*FALSE*

df_4: ( )*TRUE* ( )*FALSE*

Step 16: Using *Method User Interface*, users select df_4 is *FALSE*:

Question 1: df_3: ( )*TRUE* ( )*FALSE*

df_4: ( )*TRUE* (X)*FALSE*

Step 17: *Control Logic 1* receives df_4 =*FALSE* and performs simplification of *Map from Features to Variants*:

$$\begin{aligned} \textit{Map from Features to Variants} = \{\ & TRUE \Leftrightarrow pf_1, \\ & df_3 \Leftrightarrow pf_2, \\ & FALSE \Leftrightarrow pf_3, \\ & TRUE \Leftrightarrow pf_4, \\ & FALSE \Leftrightarrow pf_5\ \}. \end{aligned}$$

Step 18: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features = $\{df_3\}$

Set of True Features = $\{\ df_1, df_2\ \}$

Set of False Features = $\{df_4\ \}$

Step 19: *Method User Interface* updates the value of features in Q_2:

Question 2: df_3: ( )*TRUE* ( )*FALSE*

df_4: ( )*TRUE* (X)*FALSE*

Step 20: Users choose to end variant selection.

Step 21: *Method Control Logic 1* generates *Set of true process facts* $= \{pf_1, pf_4\}$. *Method Control Logic 1* sends *Set of true process facts* to *Method Control Logic 2*.

Step 22: *Method Control Logic 2* set as $TRUE$ the variants in *Set of Variants* that are bound to process facts in *Set of true process facts*. So,

Set of Variants $= \{ \; TRUE : (\{t_1, t_2, t_5\}, \{atleast(t_1), precedence(t_1{,}t_2)\})$

$pf_2 : (\{t_1, t_2, t_3\}, \emptyset)$

$pf_3 : (\{t_3, t_4, t_5\}, \{atleast(t_5), response(t_3{,}t_4)\})$

$TRUE : (\{t_1, t_2, t_3, t_4, t_5\}, \{atleast(t_5), precedence(t_2{,}t_1)\})$.

$pf_5 : (\{t_1, t_2, t_3, t_4\}, \{atleast(t_1), precedence(t_2{,}t_1)\})$

Step 23: *Method Control Logic 2* joins together all variants in *Set of Variants* that are set as $TRUE$. This is a *syntactically and semantically Correct SDL Process* to be provided to *User support framework at run time*. So,

SDL Process $= (\{t_1, t_2, t_5\}, \{atleast(t_1), precedence(t_1{,}t_2)\}) \cup$

$(\{t_1, t_2, t_3, t_4, t_5\}, \{atleast(t_5), precedence(t_1{,}t_2)\})$

SDL Process $= (\{t_1, t_2, t_3, t_4, t_5\}, \{atleast(t_1), atleast(t_5), precedence(t_1{,}t_2)\})$

Step 24: *Method Control Logic 2* finishes the selection of variants.

5.4.4.2.3 At run time

At run time, *User support framework at run time* takes the *syntactically and semantically Correct SDL Process* provided by *User support framework at configure time*. As mentioned previously, *User support framework at run time* is the same as *SDL framework at run time*. Since the operation of *SDL framework at run time* has already been described and an exemple provided (SCHAIDT; SANTOS, 2017b), no demonstration of its operation is made for the purposes of this example.

5.5 Conclusion

This paper presented SVSDL, a conceptual framework to select variants from processes modeled by Simple Declarative Language (SDL). Three main frameworks comprise SVSDL:

User support framework at design time, *User support framework at configure time*, *User support framework at run time.*

User support framework at design time provides a set of methods and data structures to enable the users to build *User support framework at design time.* The methods in *User support framework at design time* are designed to be performed in a logical sequence guaranteing that all data structures are compatible with the operation at configure time. In other words, all the data structures generated by *User support framework at design time* comply with all the necessary requirements for adequate delivery at configure time.

User support framework at configure time provides a set of methods and data structures to enable users to establish a *syntactically and semantically correct SDL process.* The methods in *User support framework at configure time* interact with each other. Each of them has its own operating mode, but at same time, they produce and update data structures that become input for the others. This is required because the procedure to select variants is user interaction dependent. The answers provided by users cannot be known in advance. Therefore, this requires a dynamic procedure to change the respective data structures as consequence of user interactions. This is most visible in *Method Logic Control 1.* This method manages all logical expression simplification procedures that define the maps from *domain facts* to *process facts.*

Variant must be syntactically correct before being merged into a configurable process model. The syntactic correctness of a variant is guaranteed by following the syntax rules of the language that models the variants in the configurable process model. In this approach, the syntactic correctness of a variant is guaranteed by complying with the SDL syntax rules (SCHAIDT; SANTOS, 2017b).

A process must be semantically correct before being merged into a configurable process model. In this approach, the semantic correctness of a variant is guaranteed by complying with the semantic rules of SDL. An SDL process is semantically correct if, and only if, it is sound, i.e., it complies with *option to complete*, *no dead task*, and *proper completion* (SCHAIDT; SANTOS, 2017b).

User support framework at run time provides a set of methods and data structures to enable users to run a *syntactically and semantically correct SDL process. User support framework at run time* is the same as *SDL framework at run time* (SCHAIDT; SANTOS, 2017b).

6 Selection of variants from PMBOK processes: an example of application of SVSDL

Abstract

In recent years, many people and organizations have become interested in project management. Studies show that the use of tools and techniques provided by reference models in project management are crucial for any organization to succeed in the business environment. A reference models can generate a set of different business processes for the same application domain. But, it is too expensive for companies to design and implement standardized business processes for each application context. Thus, an approach capable of capturing variability in process models is needed. This approach must be capable of representing families of process variants in a compact, reusable, and maintainable way. Thus, the objective of this paper is to present an example of an application in which a process generated from a reference model in project management. This paper presents two variant selection examples supported by questionnaires. The reference model is Project Management Body of Knowledge (PMBOK). Specifically, for this case, only PMBOK Project Scope Management was selected. The processes in Project Scope Management are modeled using Simple Declarative Language (SDL). SDL is a conceptual framework for modeling constraint based processes. Selection of variants is supported by Selection of Variants with Simple Declarative language (SVSDL). SVSDL is a conceptual framework to select variants from processes modeled using SDL.

Keywords: Reference models, project management, PMBOK, selection of variants, configurable process model, declarative languages.

6.1 Introduction

In recent years, many people and organizations have become interested in project management. Initially, project management concentrated on providing schedule and resource data to top management in just a few industries, such as the military and construction industries. The advance of technologies has facilitated the deployment of interdisciplinary and global work teams, which, in its turn, has changed the work environment in the world. (SCHWALBE, 2015). Nowadays, project management approaches have helped to eliminate wasted time and efforts that would have been directed at irrelevant tasks, in a rising number of industries around the world (ALOTAIBI; MAFIMISEBI, 2016).

However, experiences in the business process field have demonstrated that, if the projects are managed according to a reference model, this increases the project's chances of success (JOSLIN; MÜLLER, 2015; MIR; PINNINGTON, 2014; PSOMAS; VOUZAS; KAFETZOPOULOS, 2014). Studies show that the use of tools and techniques provided by reference models for project management are crucial for organizations to succeed in the highly competitive and continuously evolving global business environment (PARKER et al., 2013; HORNSTEIN, 2015).

The International Project Management Association (IPMA) (VUKOMANOVIĆ; YOUNG; HUYNINK, 2016), PRojects IN Controlled Environments (PRINCE2) (BENTLEY, 2015), Project Management Body of Knowledge (PMBOK) (ROSE, 2013) and Agile (PATANAKUL et al., 2016) are all examples of reference models in project management. Total Quality Management (TQM) (OAKLAND, 2014), ISO 9000 (CASTKA; CORBETT et al., 2015) and Lean Six Sigma (FURTERER, 2016) are examples of models in quality management.

Reference models in project management enable addressing the project in such a way that project management processes can be continuously improved within the organization (TENERA; PINTO, 2014; TOO; WEAVER, 2014). This is possible due the the integration that the managing reference models provides between the different areas involved in the project (NEVES et al., 2014; MARTINSUO; KILLEN, 2014; SÁNCHEZ, 2015). They highlight and promote the importance of collaboration among the different players in the project. This is also a fundamental condition for the success of the project. (OSIPOVA; ERIKSSON, 2013)

Reference models in project management have been used in companies of small, medium and large size around the world (MARCELINO-SÁDABA et al., 2014; EL-SAYEGH, 2014; REES-CALDWELL; PINNINGTON, 2013; LANDRY; MCDANIEL, 2015; CONFORTO et al., 2014). These models can also support the definition and implementation of strategic project management. This approach has been gaining popularity in recent years (PATANAKUL; SHENHAR, 2012).

Reference models in project management provide steps for delivering projects by defining the project objectives by applying facilitation tools and techniques that are provided to ensure that these activities are productively carried in project team meetings. (SIMON; CANACARI, 2012; ŞANDRU; OLARU, 2013; KERZNER, 2013; PRITCHARD; PMP et al., 2014; HELDMAN, 2013). These models, in general terms, propose to establish project teams. Project teams are responsible for running the project from beginning to end (MÜLLER; GLÜCKLER; AUBRY, 2013; MÜLLER et al., 2013).

Integration of project and product aspects also are provided for in reference models in project management. This is another important feature by making it possible to guide project stakeholders through the project's initiation, conceptual design, planning and execution phases (COHEN; ILUZ; SHTUB, 2014; JUGDEV et al., 2013). Integration of knowledge management processes across the different areas impacted by the project is also promoted by reference models. This feature is important because it fosters and enriches the project knowledge set in all its aspects, as well as facilitates dissemination (CAGLIANO; GRIMALDI; RAFELE, 2015).

Because of the increasing interest in reference models in project management, interest has also arisen in Project Management Information Systems (PMIS) (CANIËLS; BAKENS, 2012; RAYMOND; BERGERON, 2015). PMIS are software applications that help managers track projects from conception to execution. They provide them with pertinent information and collaborative tools. PMIS provide software tools for planning, scheduling, and communicating within the project activities (BRAGLIA; FROSOLINI, 2014).

PMIS must also deal with the business processes that are generated from reference models. A reference model can generate a set of different business processes for the same application domain. An application domain is a specific area of knowledge in which the reference model is being used. But, inside an application domain there are the application contexts. Reference models can provide process models for different application contexts (REICHERT; WEBER, 2012a; REICHERT; HALLERBACH; BAUER, 2015). For example, some types of processes can be reused in different application contexts with few changes in some of their components. These changes can be mandatory according to each application context (ROSA et al., 2013). However, reusing a process model in different contexts can result in a wide range of related process model variants belonging to the same process family (MILANI et al., 2016). These process variants are connected to the same business objectives and have several points in common (ROSA et al., 2013). But there are also differences due each context's specific conditions, for instance, some activities can be required for a given context, but entirely unnecessary for another (SCHUNSELAAR et al., 2014).

It is too expensive for companies to design and implement standardized business processes for each application context. (AYORA et al., 2012). This promotes a high level of interest in gathering common process knowledge for use as a process reference model, and, consequently, derive all variants in alignment with the respective application context (AYORA et al., 2013b). Thus, an approach to capture the variability in process models is needed. This approach must be capable of representing process variant families in a compact, reusable, and maintainable way. (AYORA et al., 2015).

In recent years, a number proposals have been made to deal with selecting variants in process families. In the business process management field, model-driven techniques provide diversified solutions for process variant management, i.e. for modeling, configuring, executing, and monitoring a given process family (ZHANG; HAN; OUYANG, 2014; ASSY; CHAN; GAALOUL, 2015; YONGSIRIWIT; ASSY; GAALOUL, 2016). However, most of the studies on business process variant selection, performed to date, have concentrated on imperative languages (AYORA et al., 2015). These studies presented frameworks intended to support procedures to make and select variants from configurable process models by using imperative language modeled processes. There are few studies on selecting variants using declarative languages (SCHUNSELAAR et al., 2012a) There is a dearth of studies on frameworks intended to generate configurable process models in which variants are modeled using declarative languages.

Thus, the objective of this paper is to present an example of an application in which a process is generated from a reference model in project management. This paper presents two examples of variant selection supported by questionnaires. The reference model is Project Management Body of Knowledge (PMBOK). Specifically, only PMBOK Project Scope Management was selected. The processes in Project Scope Management are modeled using Simple Declarative Language (SDL) (SCHAIDT; SANTOS, 2017b). SDL is a conceptual framework for modeling constraint based processes. Selection of variants is supported by Selection of Variants with Simple Declarative language (SVSDL) (SCHAIDT; SANTOS, 2017a). SVSDL is a conceptual framework to select variants from processes modeled using SDL.

This paper is divided into 5 sections. Section 2 introduces the main fundamentals in PMBOK Project Scope Management. Section 3 brings usage of SVSDL to design, configure and run a variant selection framework for PMBOK processes. This framework allows exactly one variant from PMBOK processes to be selected at configure time. Section 4 presents the usage of SVSDL to design, configure and run another variant selection framework for PMBOK processes. This framework allows more than one variant to be selected from PMBOK processes at configure time. Section 5 presents the conclusion of this paper.

6.2 PMBOK Project Scope Management

PMBOK consists in various activity management, monitoring and control processes. At each new project, these processes are performed in different conditions, which requires flexible modeling capacity. PMBOK in its fifth version establishes a set of 42 macro-processes in ten knowledge areas. One of these areas of knowledge is Project Scope Management which consists in five processes: Requirement Gathering, Define Scope, Create WBS, Verify Scope and Control Scope. Project Scope Management is primarily concerned with defining and controlling what is in scope for the project and what is not (ROSE, 2013). Table 24 providing an overview of the Project Scope Management processes.

These processes interact with each other and with processes in other Knowledge Areas. In the project context, the term scope can refer to Product scope or Project scope. Product scope refers to the features and functions that characterize a product, service, or result.

Tabela 24 – Project Scope Management processes

Scope Management Plan :	The process of creating a scope management plan that documents how the project scope will be defined, validated, and controlled.
Requirement Gathering:	The process of determining, documenting, and managing stakeholder needs and requirements to meet project objectives.
Define Scope:	The process of developing a detailed description of the project and its product.
Create WBS:	The process of subdividing project deliverables and project work into smaller, more manageable components.
Validate Scope:	The process of formalizing acceptance of the completed project deliverables.
Control Scope:	The process of monitoring the status of the project and product scope and managing changes to the scope baseline.

Project scope refers to the work performed to deliver a product, service, or result with the specified features and functions. The term project scope is sometimes viewed as including product scope.

The following subsections provide a brief description of each sub-process in Project Scope Management.

6.2.1 Scope Management Plan

Scope Management Plan is the process of drafting a scope management plan that documents how the project scope will be defined, validated, and controlled (ROSE, 2013). The inputs, tools and techniques, and outputs of this process are shown in Table 25.

Tabela 25 – Inputs, tools and techniques, and outputs in Scope Management Plan

Inputs:	Project management plan Project charter Enterprise environmental factors Organizational process assets
Tools and Techniques:	Expert judgment Meetings
Outputs:	Scope management plan Requirements management plan

The following subsections provide a brief description of the inputs, tools and techniques, and outputs of the Scope Management Plan.

The Project Management Plan defines the approach to be applied in planning and managing the project scope. Project Charter provides the project context required to plan the scope management processes. Enterprise environmental factors can include culture of the organization, infrastructure, personnel administration, marketplace conditions, etc. Organizational process assets can include policies and procedures and historical information as well as any lessons learned records.

Expert judgment refers to the expertise provided by a person or group of people with specialized education, knowledge, skill, experience, or training for deployment in developing scope management plans. Meetings of the project team can be convened to develop the scope management plan. These meetings can include the project manager, project sponsor, selected project team members, selected stakeholders, etc.

The scope management plan describes how the scope will be defined, developed, monitored, controlled, and verified. The requirements management plan is a component of the project management plan that describes how requirements will be analyzed, documented, and managed.

6.2.2 Requirement Gathering

Requirement Gathering is the process of determining, documenting, and managing stakeholder needs and requirements to meet project objectives (ROSE, 2013). The inputs, tools and techniques, and outputs of this process are depicted in Figure 26.

The following subsections provide a brief description of Requirement Gathering inputs, tools and techniques, and outputs.

The scope management plan provides clarity as to how project teams will determine the type of requirements that must be collected for the project. The requirements management plan provides the processes that will be used throughout the Requirement Gathering process to define and document stakeholder needs. The stakeholder management plan is used to understand stakeholder communication requirements and the level of stakeholder engagement in order to assess and adapt the level of stakeholder participation in requirements gathering activities. The project charter is used to provide the high-level description of the product, service, or result of the project so that detailed requirements can be developed. The stakeholder register is used to identify stakeholders who can provide information on the requirements.

An interview is a formal or informal approach to elicit information from stakeholders by talking to them directly. Focus groups bring together previously qualified stakeholders and subject matter experts to learn about their expectations and attitudes about a proposed product, service, or result. Facilitated workshops are focused sessions that bring

Tabela 26 – Inputs, tools and techniques, and outputs from Requirement Gathering

Inputs:	Scope management plan Requirements management plan Stakeholder management plan Project charter Stakeholder register
Tools and Techniques:	Interviews Focus groups Facilitated workshops Group creativity techniques Group decision-making techniques Questionnaires and surveys Observations Prototypes Benchmarking Context diagrams Document analysis
Outputs:	requirements documentation Requirement traceability matrix

key stakeholders together to define product requirements. A number of group activity techniques can be organized to identify project and product requirements, brainstorming is an example of group activity techniques. A group decision-making technique is an assessment process having multiple alternatives with an expected outcome in the form of future actions. Questionnaires and surveys are written sets of questions designed to quickly accumulate information from a large number of respondents. Observations provide a direct way of viewing individuals in their environment and how they perform their jobs or tasks and carry out processes. Prototyping is a method of obtaining early feedback on requirements by providing a working model of the expected product before actually building it. Benchmarking involves comparing actual or planned practices, such as processes and operations, to those of comparable organizations to identify best practices, generate ideas for improvement, and provide a basis for measuring performance. Context diagrams visually depict the product scope by showing a business system (process, equipment, computer system, etc.), and how people and other systems (actors) interact with it. Document analysis is used to elicit requirements by analyzing existing documentation and identifying information relevant to the requirements.

requirements documentation describes how individual requirements meet the project's business need. The requirements traceability matrix is a grid that links product requirements from their origin to the deliverables that satisfy them.

6.2.3 Define Scope

Define Scope is the process of developing a detailed description of the project and product (ROSE, 2013). The inputs, tools and techniques, and outputs of this process are depicted in Table 27.

Tabela 27 – Inputs, tools and techniques, and outputs from Define Scope

Inputs:	Scope management plan Project charter requirements documentation Organizational process assets
Tools and Techniques:	Expert judgment Product analysis Alternative generation Facilitated workshops
Outputs:	Project scope statement Project document updates

The scope management plan is a component of the project management plan that establishes the activities for developing, monitoring, and controlling the project scope. The project charter provides the high-level project description and product characteristics. This documentation will be used to select the requirements that will be included in the project. Organizational process assets can influence how scope is defined. Examples include policies, procedures, and templates for a project scope statement, project files from previous projects, and lessons learned from previous phases or projects. Expert judgment is often used to analyze the information needed to develop the project scope statement. For projects that have a product as a deliverable, as opposed to a service or result, product analysis can be an effective tool. Alternatives generation is a technique used to develop as many potential options as possible in order to identify different approaches to execute and perform the work of the project. Facilitated workshops consists in participation of key players with a variety of expectations and/or fields of expertise in intensive working sessions to reach a cross-functional and common understanding of the project objectives and their limits.

The project scope statement is a description of the project scope, major deliverables, assumptions, and constraints. Project documents that may be updated include stakeholder register, requirements documentation, and requirement traceability matrix.

6.2.4 Create WBS

Create WBS is the process of subdividing project deliverables and project work into smaller, more manageable components (ROSE, 2013). The inputs, tools and techniques, and outputs of this process are depicted in Table 28.

Tabela 28 – Inputs, tools and techniques, and outputs from Create WBS

Inputs:	Scope management plan Project scope statement requirements documentation Enterprise environmental factors Organizational process assets
Tools and Techniques:	Decomposition Expert judgment
Outputs:	Scope baseline Project documents updates

The following subsections provide a brief description of the inputs, tools and techniques, and outputs of Create WBS.

The scope management plan specifies how to create the WBS from the detailed project scope statement and how the WBS will be maintained and approved. The project scope statement describes the work that will be performed and the work that is excluded. Detailed requirements documentation is essential for understanding what needs to be produced as the result of the project and what needs to be done to deliver the project and its final products. Enterprise Environmental Factors are industry-specific WBS standards, relevant to the nature of the project, that serve as external reference sources for creation of the WBS. The organizational process assets are policies, procedures, and templates for the WBS, project files and lessons learned from previous projects.

Decomposition is a technique used for dividing and subdividing the project scope and project deliverables into smaller, more manageable parts. Expert judgment is often used to analyze the information needed to break down project deliverables down into smaller component parts in order to create an effective WBS.

The scope baseline is the approved version of a scope statement, work breakdown structure (WBS), and its associated WBS dictionary, that can be changed only through formal change control procedures and is used as a basis for comparison. Project documents that may be updated include, but are not limited to, requirements documentation, which may need to be updated to include approved changes.

6.2.5 Validate Scope

Validate Scope is the process of formalizing acceptance of the completed project deliverables (ROSE, 2013). The inputs, tools and techniques, and outputs of this process are depicted in Table 29.

In the following is shortly described the inputs, tools and techniques, and outputs of the

Tabela 29 – Inputs, tools and techniques, and outputs from Validate Scope

Inputs:	Project management plan Requirements documentation Requirement traceability matrix Verified deliverables Work performance data
Tools and Techniques:	Inspection Group decision-making techniques
Outputs:	Accepted deliverables Change requests Work performance information Project document updates

Validate Scope.

The project management plan contains the scope management plan and the scope baseline. The requirements documentation lists all the project, product, and other types of requirements for the project and product, along with their acceptance criteria. The requirements traceability matrix links requirements to their origin and tracks them throughout the project life cycle. Verified deliverables are project deliverables that are completed and checked for correctness through the Control Quality process. Work performance data can include the degree of compliance with requirements, number of nonconformities, severity of the nonconformities, or the number of validation cycles performed in a period of time.

Inspection includes activities such as measuring, examining, and validating to determine whether work and deliverables meet requirements and product acceptance criteria. These techniques are used to reach a conclusion when the validation is performed by the project team and other stakeholders.

Deliverables that meet the acceptance criteria are formally signed off and approved by the customer or sponsor. The change requests are processed for review and disposition through the Perform Integrated Change Control process. Work performance information includes information about project progress, such as which deliverables have started, their progress, which deliverables have finished, or which have been accepted. Project documents that may be updated as a result of the Validate Scope process include any documents that define the product or report status on product completion.

6.2.6 Control Scope

Control Scope is the process of monitoring the status of the project and product scope and managing changes to the scope baseline (ROSE, 2013). The inputs, tools and techniques, and outputs of this process are depicted in Table 30.

Tabela 30 – Inputs, tools and techniques, and outputs from Control Scope

Inputs:	Project management plan Requirements documentation Requirements traceability matrix Work performance data Organizational process assets
Tools and Techniques:	Variance analysis
Outputs:	Work performance information Change requests Project management plan updates Project documents updates Organizational process assets updates

The following subsections provide a brief description of the inputs, tools and techniques, and outputs of the Control Scope.

Requirements should be unambiguous (measurable and testable), traceable, complete, consistent, and acceptable to key stakeholders. The requirement traceability matrix helps to detect the impact of any changes or deviations from the scope baseline on the project objectives. Work performance data can include the number of change requests received, the number of requests accepted or the number of deliverables completed, etc. The organizational process assets that can influence the Control Scope process include existing formal and informal scope, control-related policies, procedures, guidelines, and monitoring and reporting methods and templates to be used.

Variance analysis is a technique for determining the cause and degree of difference between the baseline and actual performance. Project performance measurements are used to assess the magnitude of variation from the original scope baseline.

Work performance information produced includes correlated and contextualized information on how the project scope is performing compared to the scope baseline. Analysis of scope performance can result in a change request to the scope baseline or other components of the project management plan. Project management plan updates may include Scope Baseline Updates and other Baseline Updates. Project documents that may be updated include requirements documentation, and requirements traceability matrix. Organizational process assets that may be updated include causes of variances, corrective action chosen and the reasons, and other types of lessons learned from project scope control.

6.3 Approach to select variants from PMBOK processes by using SVSDL

This paper demonstrates the usage of SVSDL in selecting variants from a SDL process that is comprised of tasks from PMBOK. For this paper, the tasks selected were *Project management plan* and *Project Charter* from *Project Integration Management*. From *Project Scope Management Scope management plan*, *Requirements management plan*, *Requirements documentation*, *Requirements traceability matrix*, *Project scope statement*, *Scope baseline*, *Accepted deliverables*, *Change requests*, *Work performance information*were selected. Each task is represented by its abbreviation. The tasks and abbreviations used in this example are displayed in Table 31.

Tabela 31 – Set of tasks selected to the Examples 1 and 2

Project Management Plan	*PMP*
Project Charter	*PC*
Scope management plan	*SMP*
Requirements management plan	*RMP*
Requirement documentation	*RD*
Requirement traceability matrix	*RTM*
Project scope statement	*PSS*
Scope baseline	*SB*
Accepted deliverables	*AD*
Change requests	*CR*
Work performance information	*WPI*

SVSDL is divided in three frameworks: *Framework for user support at design time*, *Framework for user support at configure time*, and *Framework for user support at run time* (SCHAIDT; SANTOS, 2017a). Figures 57 and 58 show these frameworks.

The next two sections present two examples of application of the SVSDL framework in designing, configuring and running variants from a PMBOK process, by using the three SVSDL frameworks. First, the examples show how to use *Framework for user support at design time* to make *Framework for configure support at design time*. Second, the examples show how to use *Framework for user support at configure time* to make *Syntactically and Semantically Correct SDL Processes*. And finally, the examples show how to use *Framework for user support at run time* to run *a Syntactically and Semantically Correct SDL Process*.

6.4 First example of selection of variants with PMBOK and SVSDL

In this example, users select *Function* = *Exactly.one* at configure time. This option allows users to select only one variant at run time, no combination of variants is permitted. The

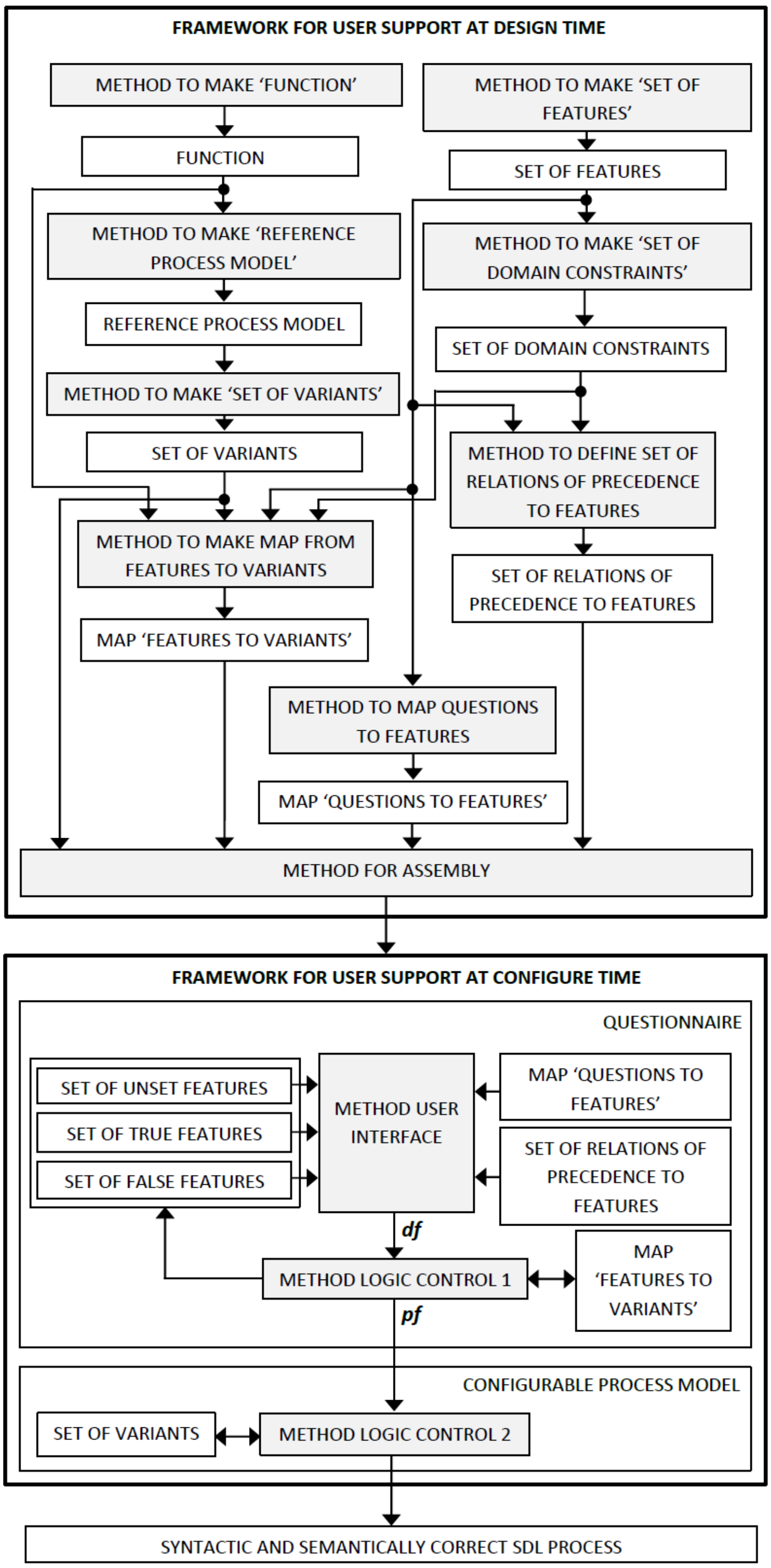

FRAMEWORK FOR USER SUPPORT AT DESIGN TIME
METHOD TO MAKE 'FUNCTION'
METHOD TO MAKE 'SET OF FEATURES'
FUNCTION
SET OF FEATURES
METHOD TO MAKE 'REFERENCE PROCESS MODEL'
METHOD TO MAKE 'SET OF DOMAIN CONSTRAINTS'
REFERENCE PROCESS MODEL
SET OF DOMAIN CONSTRAINTS
METHOD TO MAKE 'SET OF VARIANTS'
SET OF VARIANTS
METHOD TO DEFINE SET OF RELATIONS OF PRECEDENCE TO FEATURES
METHOD TO MAKE MAP FROM FEATURES TO VARIANTS
SET OF RELATIONS OF PRECEDENCE TO FEATURES
MAP 'FEATURES TO VARIANTS'
METHOD TO MAP QUESTIONS TO FEATURES
MAP 'QUESTIONS TO FEATURES'
METHOD FOR ASSEMBLY
FRAMEWORK FOR USER SUPPORT AT CONFIGURE TIME
QUESTIONNAIRE
SET OF UNSET FEATURES
SET OF TRUE FEATURES
SET OF FALSE FEATURES
METHOD USER INTERFACE
MAP 'QUESTIONS TO FEATURES'
SET OF RELATIONS OF PRECEDENCE TO FEATURES
df
METHOD LOGIC CONTROL 1
MAP 'FEATURES TO VARIANTS'
pf
CONFIGURABLE PROCESS MODEL
SET OF VARIANTS
METHOD LOGIC CONTROL 2
SYNTACTIC AND SEMANTICALLY CORRECT SDL PROCESS

Figura 57 – Framework for user support at design time and Framework for user support at configure time

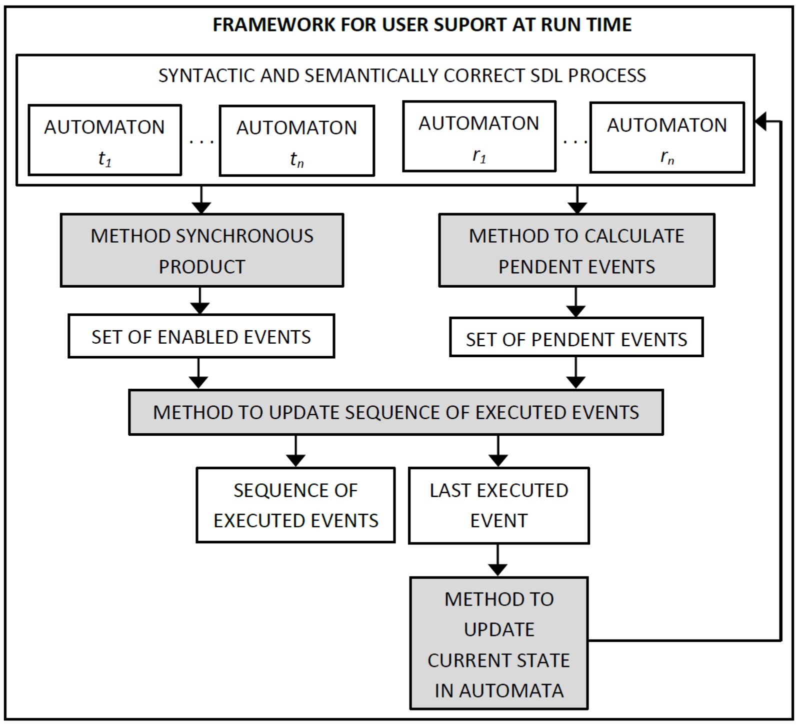

FRAMEWORK FOR USER SUPORT AT RUN TIME
SYNTACTIC AND SEMANTICALLY CORRECT SDL PROCESS
AUTOMATON t_1
AUTOMATON t_n
AUTOMATON r_1
AUTOMATON r_n
METHOD SYNCHRONOUS PRODUCT
METHOD TO CALCULATE PENDENT EVENTS
SET OF ENABLED EVENTS
SET OF PENDENT EVENTS
METHOD TO UPDATE SEQUENCE OF EXECUTED EVENTS
SEQUENCE OF EXECUTED EVENTS
LAST EXECUTED EVENT
METHOD TO UPDATE CURRENT STATE IN AUTOMATA

Figura 58 – Framework for user support at design time

following shows the SVSDL frameworks.

6.4.1 At design time

At design time, SVSDL provides users with *Framework for user support at design time*. The following section shows the sequence of steps with the different interactions between the methods comprising *Framework for user support at design time*.

Step 1: Using *Method to define Function*, users define $Function = Exactly.one$.

Step 2: Using *Method to make Reference Process Model*, users define $RPM = (T,R)$, where

$T = \{ \ \ PMP,\ PC,\ SMP,\ RMP,\ RTM,\ PSS,\ SB,\ AD,\ CR,\ WPI \ \ \}$

$R = \{ \ \ atleast1(SMP),\ atleast1(RMP),$
$precedence(SMP,\ RD),\ precedence(RD,\ AD),$
$precedence(RD,\ CR),\ precedence(SMP,\ RTM),$
$precedence(RTM,\ AD),\ precedence(RTM,\ CR),$
$precedence(RMP,\ RD),\ precedence(RMP,\ RTM),$
$response(SMP,\ RD),\ response(RD,\ AD),$
$response(RD,\ CR),\ response(SMP,\ RTM),$
$response(RTM,\ AD),\ response(RTM,\ CR),$
$response(RMP,\ RD),\ response(RMP,\ RTM) \qquad \}$

Step 3: Using *Method to make Set of Variants*, users define 16 variants:

pf_1 : ({*SMP, RD, AD*},
{*atleast1*(*SMP*), *precedence*(*SMP, RD*), *precedence*(*RD, AD*),
response(*SMP, RD*), *response*(*RD, AD*)})

pf_2 : ({*SMP, RD, AD, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*SMP*), *precedence*(*SMP, RD*), *precedence*(*RD, AD*),
response(*SMP, RD*), *response*(*RD, AD*)})

pf_3 : ({*SMP, RD, CR*},
{*atleast1*(*SMP*), *precedence*(*SMP, RD*), *precedence*(*RD, CR*),
response(*SMP, RD*), *response*(*RD, CR*)})

pf_4 : ({*SMP, RD, CR, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*SMP*), *precedence*(*SMP, RD*), *precedence*(*RD, CR*),
response(*SMP, RD*), *response*(*RD, CR*)})

pf_5 : ({*SMP, RTM, AD*},
{*atleast1*(*SMP*), *precedence*(*SMP, RTM*), *precedence*(*RTM, AD*),
response(*SMP, RTM*), *response*(*RTM, AD*)})

pf_6 : ({*SMP, RTM, AD, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*SMP*), *precedence*(*SMP, RTM*), *precedence*(*RTM, AD*),
response(*SMP, RTM*), *response*(*RTM, AD*)})

pf_7 : ({*SMP, RTM, CR*},
{*atleast1*(*SMP*), *precedence*(*SMP, RTM*), *precedence*(*RTM, CR*),
response(*SMP, RTM*), *response*(*RTM, CR*)})

pf_8 : ({*SMP, RTM, CR, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*SMP*), *precedence*(*SMP, RTM*), *precedence*(*RTM, CR*),
response(*SMP, RTM*), *response*(*RTM, CR*)})

pf_9 : ({*RMP, RD, AD*},
{*atleast1*(*RMP*), *precedence*(*RMP, RD*), *precedence*(*RD, AD*),
response(*RMP, RD*), *response*(*RD, AD*)})

pf_{10} : ({*RMP, RD, AD, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*RMP*), *precedence*(*RMP, RD*), *precedence*(*RD, AD*),
response(*RMP, RD*), *response*(*RD, AD*)})

pf_{11} : ({*RMP, RD, CR*},
{*atleast1*(*RMP*), *precedence*(*RMP, RD*), *precedence*(*RD, CR*),
response(*RMP, RD*), *response*(*RD, CR*)})

pf_{12} : ({*RMP, RD, CR, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*RMP*), *precedence*(*RMP, RD*), *precedence*(*RD, CR*),
response(*RMP, RD*), *response*(*RD, CR*)})

pf_{13} : ({*RMP, RTM, AD*},
{*atleast1*(*RMP*), *precedence*(*RMP, RTM*), *precedence*(*RTM, AD*),
response(*RMP, RTM*), *response*(*RTM, AD*)})

pf_{14} : ({*RMP, RTM, AD, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*RMP*), *precedence*(*RMP, RTM*), *precedence*(*RTM, AD*),
response(*RMP, RTM*), *response*(*RTM, AD*)})

pf_{15} : ({*RMP, RTM, CR*},
{*atleast1*(*RMP*), *precedence*(*RMP, RTM*), *precedence*(*RTM, CR*),
response(*RMP, RTM*), *response*(*RTM, CR*)})

pf_{16} : ({*RMP, RTM, CR, PMP, PC, PSS, SB, WPI*},
{*atleast1*(*RMP*), *precedence*(*RMP, RTM*), *precedence*(*RTM, CR*),
response(*RMP, RTM*), *response*(*RTM, CR*)})

Step 4: Using *Method to make Set of Features*, users define 11 features:

df_1 : *Project Management Plan*
df_2 : *Project Charter*
df_3 : *Scope management plan*
df_4 : *Requirements management plan*
df_5 : *Requirements documentation*
df_6 : *Requirements traceability matrix*
df_7 : *Project scope statement*
df_8 : *Scope baseline*
df_9 : *Accepted deliverable*
df_{10} : *Change requests*
df_{11} : *Work performance information*

Step 5: Using *Method to make Set of Domain Constraints*, users define no *domain constraint.*

Step 6: Using *Method to make Set of Relations of Precedence to Features*, users define:

Set of Relations of Precedence to Features = { *precedence*({df_1}, {df_2}),
precedence({df_2}, {df_3, df_4}),
precedence({df_3, df_4}, {df_5, df_6}),
precedence({df_5, df_6}, {df_7}),

$precedence(\{df_7\}, \{df_8\}),$
$precedence(\{df_8\}, \{df_9, df_{10}\}),$
$precedence(\{df_9, df_{10}\}, \{df_{11}\})$ }

Step 7: Using *Method to make Map from Features to Variants*, users define *Map from Features to Variants*:

Map from Features to Variants = {

pf_1: $df_1 \wedge df_2 \wedge df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_2: $df_1 \wedge df_2 \wedge df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_3: $df_1 \wedge df_2 \wedge df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_4: $df_1 \wedge df_2 \wedge df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_5: $df_1 \wedge df_2 \wedge \neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_6: $df_1 \wedge df_2 \wedge \neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_7: $df_1 \wedge df_2 \wedge \neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_8: $df_1 \wedge df_2 \wedge \neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_9: $\neg df_1 \wedge \neg df_2 \wedge df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge df_9 \wedge \neg df_{10} \wedge \neg df_{11}$,

pf_{10}: $\neg df_1 \wedge \neg df_2 \wedge df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge \neg df_9 \wedge df_{10} \wedge \neg df_{11}$,

pf_{11}: $\neg df_1 \wedge \neg df_2 \wedge df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge df_9 \wedge \neg df_{10} \wedge \neg df_{11}$,

pf_{12}: $\neg df_1 \wedge \neg df_2 \wedge df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge \neg df_9 \wedge df_{10} \wedge \neg df_{11}$,

pf_{13}: $\neg df_1 \wedge \neg df_2 \wedge \neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge df_9 \wedge \neg df_{10} \wedge \neg df_{11}$,

pf_{14}: $\neg df_1 \wedge \neg df_2 \wedge \neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge \neg df_9 \wedge df_{10} \wedge \neg df_{11}$,

pf_{15}: $\neg df_1 \wedge \neg df_2 \wedge \neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge df_9 \wedge \neg df_{10} \wedge \neg df_{11}$,

pf_{16}: $\neg df_1 \wedge \neg df_2 \wedge \neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_7 \wedge \neg df_8 \wedge \neg df_9 \wedge df_{10} \wedge \neg df_{11}$ }

Step 8: Using *Method to make Map from Questions to Features*, users define:

Map from Questions to Features = {

(Q_1: *Which documents will be developed to Develop Project Management Plan?* : $\{df_1\}$),

(Q_2: *Which documents will be developed to Develop Project Charter?* : $\{df_2\}$),

(Q_3: *Which documents will be developed to Plan Scope Management?* : $\{df_3, df_4\}$),

(Q_4: *Which documents will be developed to Requirement Gathering?* : $\{df_5, df_6\}$),

(Q_5: Which documents will be developed to Define Scope? : $\{df_7\}$),

(Q_6: Which documents will be developed to Create WBS? : $\{df_8\}$),

(Q_7: Which documents will be developed to Validate Scope? : $\{df_9, df_{10}\}$),

(Q_8: Which documents will be developed to Control Scope? : $\{df_{11}\}$) }

Step 9: From inputs defined previously, *Method for Assembly* makes *Framework for user support at configure time. Framework for user support at configure time* is used at configure time.

6.4.2 At configure time

At configure time, SVSDL provides users with *Framework for user support at configure time.* Next the sequence of steps with the different interactions between the methods that comprise *Framework for user support at configure time* is shown.

Step 1: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features.* So,

Set of Unset Features = $\{df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}\}$

Set of True Features = $\emptyset$

Set of False Features = $\emptyset$

Step 2: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features which precedent features have already been set comprise *Set of Enabled Features.* Since no feature has been set yet, the system displays

Set of Enabled Features = $\{df_1\}$.

Step 3: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* $\cap$ *Set of Enabled Features.*

Set of Enabled/Unset Features = $\{df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}\} \cap \{df_1\}$.

Set of Enabled/Unset Features = $\{df_1\}$.

Step 4: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There is only one question bound to features in *Set of Enabled/Unset Features*: *Question 1* (Q_1). So, Q_1 is displayed:

Q_1: *Will Project Management Plan be developed?*:

[ ] *Yes* ($df_1 = TRUE$) [ ] *No* ($df_1 = FALSE$)

Step 5: Using *Method User Interface*, user selects '*Yes*' ($df_1 = TRUE$):

Q_1: *Will Project Management Plan be developed?*:

[X] *Yes* ($df_1 = TRUE$) [ ] *No* ($df_1 = FALSE$)

Step 6: *Control Logic 1* receives $df_1 = TRUE$ and performs simplification of *Map from Features to Variants*. Since $df_1 = TRUE$, pf_3 and pf_4 are automatically *FALSE*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $df_2 \wedge df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_2: $df_2 \wedge df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_3: $df_2 \wedge df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_4: $df_2 \wedge df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_5: $df_2 \wedge \neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_6: $df_2 \wedge \neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$,

pf_7: $df_2 \wedge \neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge df_9 \wedge \neg df_{10} \wedge df_{11}$,

pf_8: $df_2 \wedge \neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge df_7 \wedge df_8 \wedge \neg df_9 \wedge df_{10} \wedge df_{11}$

pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

But simplification must ensure that at least one variant is selected, so df_2, df_3, df_7, df_8 are $TRUE$ because they are in all conjunctions. Thus,

Map from Features to Variants = {

pf_1: $df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_2: $df_3 \wedge \neg df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_3: $df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_4: $df_3 \wedge \neg df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_5: $\neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_6: $\neg df_3 \wedge df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_7: $\neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_8: $\neg df_3 \wedge df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

Step 7: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features.* So,

Set of Unset Features = {df_3, df_4, df_5, df_6, df_9, df_{10}}

Set of True Features = {df_1, df_2, df_7, df_8, df_{11}}

Set of False Features = $\emptyset$

Step 8: *Method User Interface* updates the value of features in Q_1:

Q_1: *Will Project Management Plan be developed?*:

[X] *Yes* ($df_1 = TRUE$) [ ] *No* ($df_1 = FALSE$)

Step 9: Since users cannot set any other feature in Q_1, users choose to close Q_1.

Step 10: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features for which precedent features have already been set comprise *Set of Enabled Features.* So, the system returns

Set of Enabled Features = {df_1, df_2, df_3, df_4, df_8, df_9, df_{10}}.

Step 11: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* $\cap$ *Set of Enabled Features.*

Set of Enabled/Unset Features = {df_3, df_4, df_5, df_6, df_9, df_{10}} $\cap$ {df_2, df_3, df_4, df_8, df_9, df_{10}}.

Set of Enabled/Unset Features = {df_3, df_4, df_9, df_{10}}.

Step 12: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There are two questions bound to features in *Set of Enabled/Unset Features*: *Question 3* (Q_3) and

Question 7 (Q_7). Users choose to answer Q_3. So, Q_3 is presented to the user:

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3) [ ] *Yes* [ ] *No*
- *Requirements management plan* (df_4) [ ] *Yes* [ ] *No*

Step 13: Using *Method User Interface*, user selects df_3 = '*No*' (df_3 = *FALSE*):

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3) [ ] *Yes* [X] *No*
- *Requirements management plan* (df_4) [ ] *Yes* [ ] *No*

Step 14: *Control Logic 1* receives $df_1 = TRUE$ and performs simplification of *Map from Features to Variants*. Since $df_1 = TRUE$, pf_3 and pf_4 are automatically *FALSE*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $FALSE$, pf_3: $FALSE$, pf_4: $FALSE$,

pf_5: $df_4 \wedge df_5 \wedge \neg df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_6: $df_4 \wedge df_5 \wedge \neg df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_7: $df_4 \wedge \neg df_5 \wedge df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_8: $df_4 \wedge \neg df_5 \wedge df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

But simplification must ensure that at least one variant is selected, so df_4 is $TRUE$ because it is in all conjunctions. Thus,

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $FALSE$, pf_3: $FALSE$, pf_4: $FALSE$,

pf_5: $df_5 \wedge \neg df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_6: $df_5 \wedge \neg df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_7: $\neg df_5 \wedge df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_8: $\neg df_5 \wedge df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

Step 15: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features = $\{df_5, df_6, df_9, df_{10}\}$

Set of True Features = $\{df_1, df_2, df_4, df_7, df_8, df_{11}\}$

Set of False Features = $\{df_3\}$

Step 16: *Method User Interface* updates the value of features in Q_3:

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3) [ ] *Yes* [X] *No*
- *Requirements management plan* (df_4) [X] *Yes* [ ] *No*

Step 17: Since users cannot set any other feature in Q_3, users choose to close Q_3.

Step 17: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features for which precedent features have already been set comprise *Set of Enabled Features.* So, the system returns

Set of Enabled Features = $\{df_1, df_2, df_3, df_4, df_5, df_6, df_8, df_9, df_{10}\}$.

Step 19: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Enabled Features* ∩ *Set of Unset Features.*

Set of Enabled/Unset Features = $\{df_5, df_6, df_9, df_{10}\} \cap \{df_1, df_2, df_3, df_4, df_5, df_6, df_8, df_9, df_{10}\}$.

Set of Enabled/Unset Features = $\{df_5, df_6, df_9, df_{10}\}$.

Step 20: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There are two questions bound to features in *Set of Enabled/Unset Features*: *Question 4* (Q_4) and *Question 7* (Q_7). Users choose to answer Q_4. So, Q_4 is displayed to users:

Q_4: *Which documents will be developed for Requirement Gathering?*

- *Requirements documentation* (df_5) [ ] *Yes* [ ] *No*
- *Requirements traceability matrix* (df_6) [ ] *Yes* [ ] *No*

Step 21: Using *Method User Interface*, user selects df_5 = '*Yes*' (df_5 = *TRUE*):

Q_4: *Which documents will be developed for Requirement Gathering?*

- *Requirements documentation* (df_5) [X] *Yes* [ ] *No*
- *Requirements traceability matrix* (df_6) [ ] *Yes* [ ] *No*

Step 22: *Control Logic 1* receives $df_5 = TRUE$ and performs simplification of *Map from Features to Variants*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $FALSE$, pf_3: $FALSE$, pf_4: $FALSE$,

pf_5: $\neg df_6 \wedge df_9 \wedge \neg df_{10}$,

pf_6: $\neg df_6 \wedge \neg df_9 \wedge df_{10}$,

pf_7: $FALSE$, pf_8: $FALSE$, pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$,
pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

But simplification must ensure that at least one variant is selected, so df_6 is $FALSE$ because it is in all conjunctions. Thus,

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $FALSE$, pf_3: $FALSE$, pf_4: $FALSE$,

pf_5: $df_9 \wedge \neg df_{10}$,

pf_6: $\neg df_9 \wedge df_{10}$,

pf_7: $FALSE$, pf_8: $FALSE$, pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$,
pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

Step 23: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features = {df_9, df_{10}}

Set of True Features = {df_1, df_2, df_4, df_5, df_7, df_8, df_{11}}

Set of False Features = {df_3, df_6}

Step 24: *Method User Interface* updates the value of features in Q_4:

- *Requirements documentation* (df_5) [X] *Yes* [ ] *No*
- *Requirements traceability matrix* (df_6) [ ] *Yes* [X] *No*

Step 25: Since users cannot set any other feature in Q_4, users choose to close Q_4.

Step 26: *Method User Interface* checks *Set of Relations of Precedence to Features*. From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features for which precedent features have already been set comprise *Set of Enabled Features*. So, the system returns

Set of Enabled Features = {df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}}.

Step 27: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* ∩ *Set of Enabled Features.*

Set of Enabled/Unset Features = {df_9, df_{10}} ∩ {df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}}.

Set of Enabled/Unset Features = {df_9, df_{10}}.

Step 28: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There is one questions bound to features in *Set of Enabled/Unset Features*: *Question 7* (Q_7). So, Q_7 is presented to users:

Q_7: Which documents will be developed to Validate Scope?

- *Scope management plan* (df_9) [ ] *Yes* [ ] *No*
- *Requirements management plan* (df_{10}) [ ] *Yes* [ ] *No*

Step 29: Using *Method User Interface*, user selects df_9 = '*Yes*' (df_9 = *TRUE*):

Q_7: Which documents will be developed to Validate Scope?

- *Scope management plan* (df_9) [X] *Yes* [ ] *No*
- *Requirements management plan* (df_{10}) [ ] *Yes* [ ] *No*

Step 30: *Control Logic 1* receives df_9 =*TRUE* and performs simplification of *Map from Features to Variants*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $FALSE$, pf_3: $FALSE$, pf_4: $FALSE$,

pf_5: $\neg df_{10}$,

pf_6: $FALSE$, pf_7: $FALSE$, pf_8: $FALSE$, pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

But simplification must ensure that at least one variant be selected, so df_{10} is $FALSE$ because it is in all conjunctions. Thus,

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $FALSE$, pf_3: $FALSE$, pf_4: $FALSE$, pf_5: $TRUE$, pf_6: $FALSE$, pf_7: $FALSE$, pf_8: $FALSE$, pf_9: $FALSE$, pf_{10}: $FALSE$, pf_{11}: $FALSE$, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $FALSE$, pf_{15}: $FALSE$, pf_{16}: $FALSE$ }

Step 31: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features.* So,

Set of Unset Features = $\emptyset$

Set of True Features = $\{df_1, df_2, df_4, df_5, df_7, df_8, df_9, df_{11}\}$

Set of False Features = $\{df_3, df_6, df_{10}\}$

Step 32: *Method User Interface* updates the value of features in Q_7:

Q_7: Which documents will be developed to Validate Scope?

- *Scope management plan* (df_9) [X] *Yes* [ ] *No*
- *Requirements management plan* (df_{10}) [ ] *Yes* [X] *No*

Step 33: Since users cannot set any other feature in Q_7, users choose to close Q_4.

Step 34: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features for which precedent features have already been set comprise *Set of Enabled Features.* So, the system returns

Set of Enabled Features = $\{df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}\}$.

Step 35: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* $\cap$ *Set of Enabled Features.*

Set of Enabled/Unset Features = $\emptyset \cap \{df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}\}$.

Set of Enabled/Unset Features = $\emptyset$.

Step 36: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. Since *Set of Enabled/Unset Features* $= \emptyset$, there is no question to be presented to the user.

Step 37: Users choose to finish selection of variants.

Step 38: *Method Control Logic 1* makes *Set of true process facts* $= \{pf_5\}$ and sends it to *Method Control Logic 2*.

Step 39: *Method Control Logic 2* set as $TRUE$ the variants in *Set of Variants* that is bound to *process facts* in *Set of true process facts*. Since *Set of true process facts* $= \{pf_5\}$, the system returns

$TRUE$: ({*SMP, RTM, AD*},
{*atleast1*(*SMP*), *precedence*(*SMP, RTM*), *precedence*(*RTM, AD*),
response(*SMP, RTM*), *response*(*RTM, AD*)})

Step 40: *Method Control Logic 2* joins all variants in *Set of Variants* that are set as $TRUE$. This is *a Syntactically and Semantically Correct SDL Process* to be provided to *Framework for User Support at Run Time*. So,

SDL Process = ({*SMP, RTM, AD*},
{*atleast1*(*SMP*), *precedence*(*SMP, RTM*), *precedence*(*RTM, AD*),
response(*SMP, RTM*), *response*(*RTM, AD*)})

Step 41: *Method Control Logic 2* finishes the selection of variants.

6.4.3 At run time

At run time, SVSDL provides users with *Framework for user support at run time*. The following section presents the sequence of steps with the different interactions among the methods comprising *Framework for user support at run time*.

Step 1: *Framework for user support at run time* creates the automaton for each task and constraint taken from *Framework for user support at configure time*. These automata are shown in Figure 59. These automata are applied in making the sequence of markings that is presented in Figure .

Step 2: *Framework for user support at run time* calculates marking M = 111111.

Step 3: *Method Synchronous product* makes *Set of enabled events* $= \{PC(s), SMP(s)\}$.

Step 4: *Method to calculate pending events* makes *Set of pending events* $= \{PC(c),$

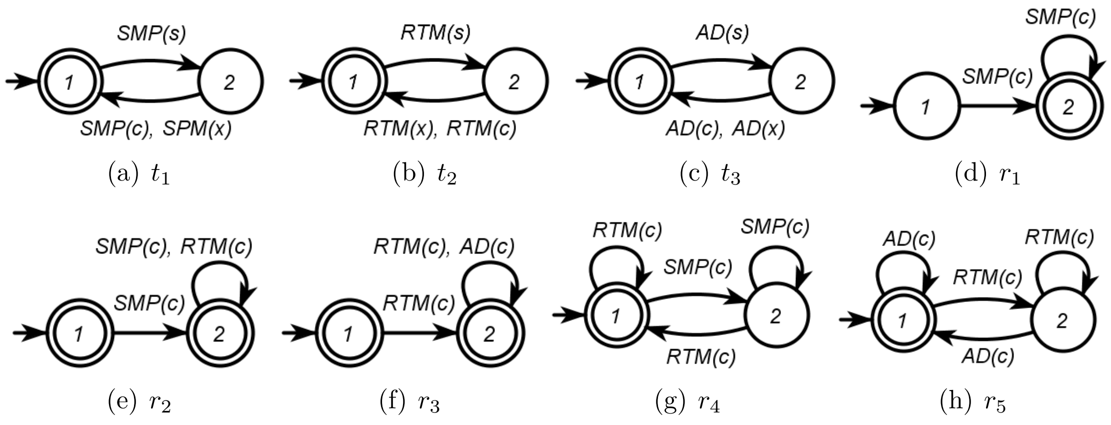


Figura 59 – a Syntactically and Semantically SDL process taken from configure time

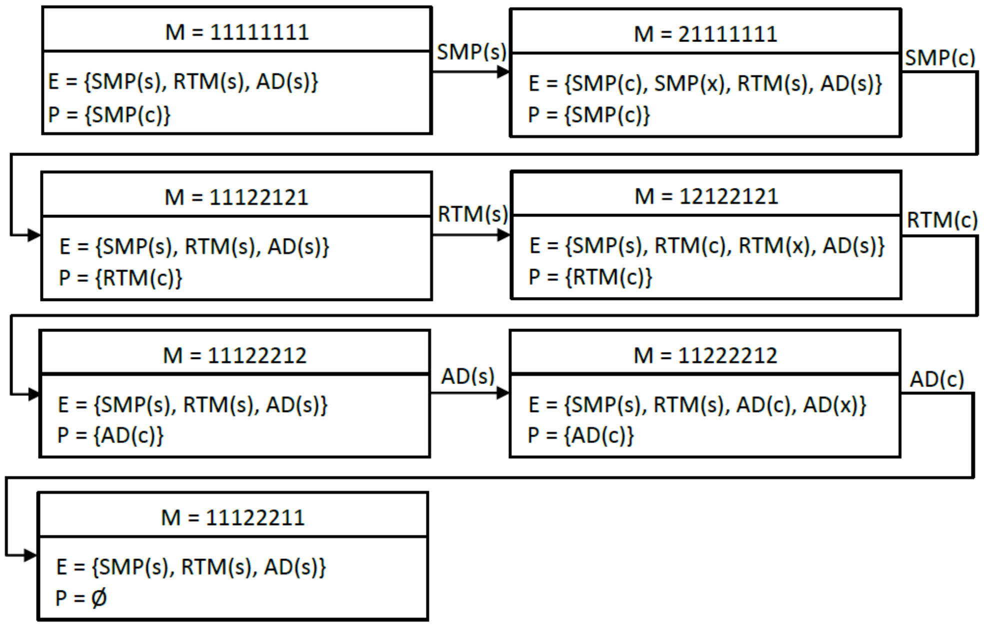

M = 11111111
E = {SMP(s), RTM(s), AD(s)}
P = {SMP(c)}
SMP(s)
M = 21111111
E = {SMP(c), SMP(x), RTM(s), AD(s)}
P = {SMP(c)}
SMP(c)
M = 11122121
E = {SMP(s), RTM(s), AD(s)}
P = {RTM(c)}
RTM(s)
M = 12122121
E = {SMP(s), RTM(c), RTM(x), AD(s)}
P = {RTM(c)}
RTM(c)
M = 11122212
E = {SMP(s), RTM(s), AD(s)}
P = {AD(c)}
AD(s)
M = 11222212
E = {SMP(s), RTM(s), AD(c), AD(x)}
P = {AD(c)}
AD(c)
M = 11122211
E = {SMP(s), RTM(s), AD(s)}
P = Ø

Figura 60 – Sequence of markings

$SMP(c)\}$.

Step 5: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to users.

Step 6: Users choose to execute $PC(s)$.

Step 7: *Method to update sequence of executed events* updates *Set of executed events* = *Sequence of executed events* = $PC(s)$.

Step 8: *Method to update current state in automata* takes the last executed event from *Sequence of executed events* to update the current state in each automaton of the SDL process.

Step 9: *Framework for user support at run time* calculates marking M = 121111.

Step 10: *Method Synchronous product* makes *Set of enabled events* = $\{PC(c)$, $PC(x)$, $SMP(s)\}$.

Step 11: *Method to calculate pending events* makes *Set of pending events* = $\{PC(c)$, $SMP(c)\}$.

Step 12: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Step 13: Users choose to execute $PC(c)$.

Step 14: *Method to update sequence of executed events* takes $PC(c)$ and updates *Sequence of executed events* = $PC(s).PC(c)$

Step 15: *Method to update current state in automata* takes $PC(c)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 16: *Framework for user support at run time* calculates marking M = 111222.

Step 17: *Method Synchronous product* makes *Set of enabled events* = $\{PC(s)$, $SMP(s)\}$.

Step 18: *Method to calculate pending events* makes *Set of pending events* = $\{SMP(c)\}$.

Step 19: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Step 20: Users choose to execute $SMP(s)$.

Step 21: *Method to update sequence of executed events* takes $SMP(s)$ and updates *Sequence of executed events* = $PC(s).PC(c).SMP(s)$

Step 22: *Method to update current state in automata* takes $SMP(s)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 23: *Framework for user support at run time* calculates marking M = 211222.

Step 24: *Method Synchronous product* makes *Set of enabled events* = $\{PC(s)$, $SMP(c)$, $SMP(x)\}$.

Step 25: *Method to calculate pending events* makes *Set of pending events* = $\{SMP(c)\}$.

Step 26: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Step 27: Users choose to execute $SMP(c)$.

Step 28: *Method to update sequence of executed events* takes $SMP(c)$ and updates *Sequence of executed events* = $PC(s).PC(c).SMP(s).SMP(c)$

Step 29: *Method to update current state in automata* takes $SMP(c)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 30: *Framework for user support at run time* calculates marking M = 112221.

Step 31: *Method Synchronous product* makes *Set of enabled events* = $\{PC(s), SMP(s)\}$.

Step 32: *Method to calculate pending events* makes *Set of pending events* = $\emptyset$.

Step 33: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Since no event is pending, users are able to continue or finish the running process, from step 33.

6.5 Second example of selection of variants with PMBOK and SVSDL

This section demonstrates another use of SVSDL to select variants from an SDL process comprised of PMBOK tasks. For this example, the same tasks in Table 31 were selected .

This section presents the use of the three frameworks that comprise SVSDL: *Framework for user support at design time*, *Framework for user support at configure time*, and *Framework for user support at run time.* In this application example, users select *Function* = *Atleast.one* at configure time. This option allows users to select more than one variant at run time, a combination of variants is permitted. The following section presents SVSDL frameworks.

6.5.1 At design time

At design time, SVSDL provides users with *Framework for user support at design time.* The sequence of steps with the interactions among the methods comprising *Framework for user support at design time* is shown.

Step 1: Using *Method to define Function*, users define *Function* = *Atleast.one.*

Step 2: Using *Method to make Reference Process Model*, users define $RPM = (T,R)$, where

T = {*PMP, PC, SMP, RMP, RTM, PSS, SB, AD, CR, WPI*}

R = { *atleast1*(*PMP*), *atleast1*(*PC*), *atleast1*(*SMP*),

atleast1(*RMP*), *atleast1*(*RD*), *atleast1*(*RTM*),

atleast1(*PSS*), *atleast1*(*SB*), *atleast1*(*AD*),

atleast1(*CR*), *atleast1*(*WPI*),

precedence(*PMP, SMP*), *response*(*PMP, SMP*),

precedence(*PMP, RMP*), *response*(*PMP, RMP*),

precedence(*PC, SMP*), *response*(*PC, SMP*),

precedence(*PC, RMP*), *response*(*PC, RMP*),

precedence(*SMP, RD*), *response*(*SMP, RD*),

precedence(*SMP, RTM*), *response*(*SMP, RTM*),

precedence(*RMP, RD*), *response*(*RMP, RD*),

precedence(*RMP, RTM*), *response*(*RMP, RTM*),

precedence(*SMP, PSS*), *response*(*SMP, PSS*),

precedence(*RMP, PSS*), *response*(*RMP, PSS*),

precedence(*SMP, SB*), *response*(*SMP, SB*),

precedence(*RMP, SB*), *response*(*RMP, SB*),

precedence(*RD, PSS*), *response*(*RD, PSS*),

precedence(*RTM, PSS*), *response*(*RTM, PSS*),

precedence(*RD, AD*), *response*(*RD, AD*),

precedence(*RD, CR*), *response*(*RD, CR*),

precedence(*RTM, AD*), *response*(*RTM, AD*),

precedence(*RTM, CR*), *response*(*RTM, CR*),

precedence(*RD, WPI*), *response*(*RD, WPI*),

precedence(*RTM, WPI*), *response*(*RTM, WPI*),

precedence(*PSS, SB*), *response*(*PSS, SB*)}

Step 3: Using *Method to make Set of Variants*, users define 32 variants:

Set of Variants = {

pf_1: ({*PMP*}, {*atleast1*(*PMP*)}),

pf_2: ({*PC*}, {*atleast1*(*PC*)}),

pf_3: ({*SMP*}, {*atleast1*(*SMP*)}),

pf_4: ({*RMP*}, {*atleast1*(*RMP*)}),

pf_5: ({*RD*}, {*atleast1*(*RD*)}),

pf_6: ({*RTM*}, {*atleast1*(*RTM*)}),

pf_7: ({*PSS*}, {*atleast1*(*PSS*)}),

pf_8: ({*SB*}, {*atleast1*(*SB*)}),

pf_9: ({*AD*}, {*atleast1*(*AD*)}),

pf_{10}: ({*CR*}, {*atleast1*(*CR*)}),

pf_{11}: ({*WPI*}, {*atleast1*(*WPI*)}),

pf_{12}: ({*PMP, SMP*}, {*precedence*(*PMP, SMP*), *response*(*PMP, SMP*)}),

pf_{13}: ({*PMP, RMP*}, {*precedence*(*PMP, RMP*), *response*(*PMP, RMP*)}),

pf_{14}: ({*PC, SMP*}, {*precedence*(*PC, SMP*), *response*(*PC, SMP*)}),

pf_{15}: ({*PC, RMP*}, {*precedence*(*PC, RMP*), *response*(*PC, RMP*)}),

pf_{16}: ({*SMP, RD*}, {*precedence*(*SMP, RD*), *response*(*SMP, RD*)}),

pf_{17}: ({*SMP, RTM*}, {*precedence*(*SMP, RTM*), *response*(*SMP, RTM*)}),

pf_{18}: ({*RMP, RD*}, {*precedence*(*RMP, RD*), *response*(*RMP, RD*)}),

pf_{19}: ({*RMP, RTM*}, {*precedence*(*RMP, RTM*), *response*(*RMP, RTM*)}),

pf_{20}: ({*SMP, PSS*}, {*precedence*(*SMP, PSS*), *response*(*SMP, PSS*)}),

pf_{21}: ({*RMP, PSS*}, {*precedence*(*RMP, PSS*), *response*(*RMP, PSS*)}),

pf_{22}: ({*SMP, SB*}, {*precedence*(*SMP, SB*), *response*(*SMP, SB*)}),

pf_{23}: ({*RMP, SB*}, {*precedence*(*RMP, SB*), *response*(*RMP, SB*)}),

pf_{24}: ({*RD, PSS*}, {*precedence*(*RD, PSS*), *response*(*RD, PSS*)}),

pf_{25}: ({*RTM, PSS*}, {*precedence*(*RTM, PSS*), *response*(*RTM, PSS*)}),

pf_{26}: ({*RD, AD*}, {*precedence*(*RD, AD*), *response*(*RD, AD*)}),

pf_{27}: ({*RD, CR*}, {*precedence*(*RD, CR*), *response*(*RD, CR*)}),

pf_{28}: ({*RTM, AD*}, {*precedence*(*RTM, AD*), *response*(*RTM, AD*)}),

pf_{29}: ({*RTM, CR*}, {*precedence*(*RTM, CR*), *response*(*RTM, CR*)}),

pf_{30}: ({*RD, WPI*}, {*precedence*(*RD, WPI*), *response*(*RD, WPI*)}),

pf_{31}: ({*RTM, WPI*}, {*precedence*(*RTM, WPI*), *response*(*RTM, WPI*)}),

pf_{32}: ({*PSS, SB*}, {*precedence*(*PSS, SB*), *response*(*PSS, SB*)}) }

Step 4: Using *Method to make Set of Features*, users define 11 features:

df_1 : *Project Management Plan*
df_2 : *Project Charter*
df_3 : *Scope management plan*

df_4 : *Requirements management plan*
df_5 : *Requirements documentation*
df_6 : *Requirements traceability matrix*
df_7 : *Project scope statement*
df_8 : *Scope baseline*
df_9 : *Accepted deliverable*
df_{10} : *Change requests*
df_{11} : *Work performance information*

Step 5: Using *Method to make Set of Domain Constraints*, users define no *domain constraint.*

Step 6: Using *Method to make Set of Relations of Precedence to Features*, users define:

Set of Relations of Precedence to Features = { *precedence*({df_1}, {df_2}),
precedence({df_2}, {df_3, df_4}),
precedence({df_3, df_4}, {df_5, df_6}),
precedence({df_5, df_6}, {df_7}),
precedence({df_7}, {df_8}),
precedence({df_8}, {df_9, df_{10}}),
precedence({df_9, df_{10}}, {df_{11}}) }

Step 7: Using *Method to make Map from Features to Variants*, users define *Map from Features to Variants*:

Map from Features to Variants = {

pf_1: df_1, pf_2: df_2, pf_3: df_3, pf_4: df_4, pf_5: df_5, pf_6: df_6, pf_7: df_7, pf_8: df_8,

pf_9: df_9, pf_{10}: df_{10}, pf_{11}: df_{11}, pf_{12}: $df_1 \wedge df_3$, pf_{13}: $df_1 \wedge df_4$, pf_{14}: $df_2 \wedge df_3$,

pf_{15}: $df_2 \wedge df_4$, pf_{16}: $df_3 \wedge df_5$, pf_{17}: $df_3 \wedge df_6$, df_{18}: $df_4 \wedge df_5$, df_{19}: $df_4 \wedge df_6$,

pf_{20}: $df_3 \wedge df_7$, pf_{21}: $df_4 \wedge df_7$, pf_{22}: $df_3 \wedge df_8$, pf_{23}: $df_4 \wedge df_8$, pf_{24}: $df_5 \wedge df_7$,

pf_{25}: $df_6 \wedge df_7$, pf_{26}: $df_5 \wedge df_9$, pf_{27}: $df_5 \wedge df_{10}$, pf_{28}: $df_6 \wedge df_9$, pf_{29}: $df_6 \wedge df_{10}$,

pf_{30}: $df_5 \wedge df_{11}$, pf_{31}: $df_6 \wedge df_{11}$, pf_{32}: $df_7 \wedge df_8$ }

Step 8: Using *Method to make Map from Questions to Features*, users define:

Map from Questions to Features = {

(Q_1: *Which documents will be developed to Develop Project Management Plan?* : {df_1}),

(Q_2: *Which documents will be developed to Develop Project Charter?* : {df_2}),

(Q_3: *Which documents will be developed to Plan Scope Management?* : {df_3, df_4}),

(Q_4: *Which documents will be developed to Requirement Gathering?* : {df_5, df_6}),

(Q_5: Which documents will be developed to Define Scope? : {df_7}),

(Q_6: Which documents will be developed to Create WBS? : {df_8}),

(Q_7: Which documents will be developed to Validate Scope? : {df_9, df_{10}}),

(Q_8: Which documents will be developed to Control Scope? : {df_{11}}) }

Step 9: From previously defined inputs, *Method for Assembly* makes *Framework for user support at configure time. Framework for user support at configure time* is used at configure time.

6.5.2 At configure time

At configure time, SVSDL provide users with *Framework for user support at configure time.* The sequence of steps with the different interactions among the methods comprising *Framework for user support at configure time* is shown.

Step 1: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features.* So,

Set of Unset Features = {df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11} }

Set of True Features = $\emptyset$

Set of False Features = $\emptyset$

Step 2: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features comprise *Set of Enabled Features.* Since no feature has been set yet, the system returns

Set of Enabled Features = {df_1}.

Step 3: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* $\cap$ *Set of Enabled Features.*

Set of Enabled/Unset Features = {df_1, df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}} $\cap$ {df_1}.

Set of Enabled/Unset Features = {df_1}.

Step 4: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There is only one question bound to features in *Set oF Enabled/Unset Features*: *Question 1* (Q_1). So, Q_1 is presented to the user:

Q_1: *Will Project Management Plan be developed?*:

[ ] *Yes* ($df_1 = TRUE$) [ ]*No* ($df_1 = FALSE$)

Step 5: Using *Method User Interface*, users select '*No*' ($df_1 = FALSE$):

Q_1: *Will Project Management Plan be developed?*:

[ ] *Yes* ($df_1 = TRUE$) [X]*No* ($df_1 = FALSE$)

Step 6: *Control Logic 1* receives df_1 =*FALSE* and performs simplification of *Map from Features to Variants*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: df_2, pf_3: df_3, pf_4: df_4, pf_5: df_5, pf_6: df_6, pf_7: df_7,

pf_8: df_8, pf_9: df_9, pf_{10}: df_{10}, pf_{11}: df_{11}, pf_{12}: $FALSE$, pf_{13}: $FALSE$,

pf_{14}: $df_2 \wedge df_3$, pf_{15}: $df_2 \wedge df_4$, pf_{16}: $df_3 \wedge df_5$, pf_{17}: $df_3 \wedge df_6$, pf_{18}: $df_4 \wedge df_5$,

pf_{19}: $df_4 \wedge df_6$, pf_{20}: $df_3 \wedge df_7$, pf_{21}: $df_4 \wedge df_7$, pf_{22}: $df_3 \wedge df_8$, pf_{23}: $df_4 \wedge df_8$,

pf_{24}: $df_5 \wedge df_7$, pf_{25}: $df_6 \wedge df_7$, pf_{26}: $df_5 \wedge df_9$, pf_{27}: $df_5 \wedge df_{10}$, pf_{28}: $df_6 \wedge df_9$,

pf_{29}: $df_6 \wedge df_{10}$, pf_{30}: $df_5 \wedge df_{11}$, pf_{31}: $df_6 \wedge df_{11}$, pf_{32}: $df_7 \wedge df_8$ }

Step 7: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features = {df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}}

Set of True Features = $\emptyset$

Set of False Features = { df_1 }

Step 8: *Method User Interface* updates the value of features in Q_1:

Q_1: *Will Project Management Plan be developed?*:

[ ] *Yes* ($df_1 = TRUE$) [X]*No* ($df_1 = FALSE$)

Step 9: Since users cannot set any other feature in Q_1, users choose to close Q_1.

Step 10: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features for which precedent features have already been set comprise *Set of Enabled Features.* So, the system returns

Set of Enabled Features = $\{df_2\}$.

Step 11: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* $\cap$ *Set of Enabled Features.*

Set of Enabled/Unset Features = $\{df_2, df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}\} \cap \{df_2\}$.

Set of Enabled/Unset Features = $\{df_2\}$.

Step 12: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There is one question bound to the features in *Set of Enabled/Unset Features*: *Question 2* (Q_2). So, Q_2 is presented to users:

(Q_2: *Will Develop Project Charter be developed?*

[ ] *Yes* ($df_2 = TRUE$) [ ] *No* ($df_2 = FALSE$)

Step 13: Using *Method User Interface*, users select df_2 = '*No*' (df_2 = *FALSE*):

(Q_2: *Will Develop Project Charter be developed?*

[X] *Yes* ($df_2 = TRUE$) [ ] *No* ($df_2 = FALSE$)

Step 14: *Control Logic 1* receives $df_2 = TRUE$ and performs simplification of *Map from Features to Variants.* So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $TRUE$, pf_3: df_3, pf_4: df_4, pf_5: df_5, pf_6: df_6, pf_7: df_7, pf_8: df_8, pf_9: df_9, pf_{10}: df_{10}, pf_{11}: df_{11}, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: df_3, pf_{15}: df_4, pf_{16}: $df_3 \wedge df_5$, pf_{17}: $df_3 \wedge df_6$, pf_{18}: $df_4 \wedge df_5$, pf_{19}: $df_4 \wedge df_6$, pf_{20}: $df_3 \wedge df_7$, pf_{21}: $df_4 \wedge df_7$, pf_{22}: $df_3 \wedge df_8$, pf_{23}: $df_4 \wedge df_8$, pf_{24}: $df_5 \wedge df_7$, pf_{25}: $df_6 \wedge df_7$, pf_{26}: $df_5 \wedge df_9$, pf_{27}: $df_5 \wedge df_{10}$, pf_{28}: $df_6 \wedge df_9$, pf_{29}: $df_6 \wedge df_{10}$, pf_{30}: $df_5 \wedge df_{11}$, pf_{31}: $df_6 \wedge df_{11}$, pf_{32}: $df_7 \wedge df_8$ }

Step 15: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of*

False Features. So,

Set of Unset Features = { df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11} }

Set of True Features = { df_2 }

Set of False Features = { df_1 }

Step 16: *Method User Interface* updates the value of features in Q_2:

(Q_2: *Will Develop Project Charter be developed?*

[X] *Yes* ($df_2 = TRUE$)   [ ] *No* ($df_2 = FALSE$)

Step 17: Since users cannot set any other feature in Q_2, users choose to close Q_2.

Step 18: *Method User Interface* checks *Set of Relations of Precedence to Features.* From *Set of Relations of Precedence to Features*, *Method User Interface* defines the features for which precedent features have already been set. These features for which precedent features have already been set comprise *Set of Enabled Features.* So, the system returns

Set of Enabled Features = {df_3, df_4}.

Step 19: *Method User Interface* makes

Set of Enabled/Unset Features = *Set of Unset Features* ∩ *Set of Enabled Features.*

Set of Enabled/Unset Features = {df_3, df_4, df_5, df_6, df_7, df_8, df_9, df_{10}, df_{11}} ∩ {df_3, df_4}.

Set of Enabled/Unset Features = {df_3, df_4}.

Step 20: *Method User Interface* checks *Set of Enabled/Unset Features* and *Map from Questions to Features* to define the questions that can be presented to users. There is one question bound to the features in *Set of Enabled/Unset Features*: *Question 3* (Q_3). So, Q_3 is presented to users:

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3)   [ ] *Yes*   [ ] *No*
- *Requirements management plan* (df_4) [ ] *Yes*   [ ] *No*

Step 21: Using *Method User Interface*, user selects df_3 = '*Yes*' ($df_3 = TRUE$):

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3)   [X] *Yes*   [ ] *No*

- *Requirements management plan* (df_4) [ ] *Yes* [ ] *No*

Step 22: *Control Logic 1* receives df_3 =*TRUE* and performs simplification of *Map from Features to Variants*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $TRUE$, pf_3: $TRUE$, pf_4: df_4, pf_5: df_5, pf_6: df_6, pf_7: df_7,

pf_8: df_8, pf_9: df_9, pf_{10}: df_{10}, pf_{11}: df_{11}, pf_{12}: $FALSE$, pf_{13}: $FALSE$,

pf_{14}: $TRUE$, pf_{15}: df_4, pf_{16}: df_5, pf_{17}: df_6, pf_{18}: $df_4 \wedge df_5$,

pf_{19}: $df_4 \wedge df_6$, pf_{20}: $df_3 \wedge df_7$, pf_{21}: $df_4 \wedge df_7$, pf_{22}: $df_3 \wedge df_8$, pf_{23}: $df_4 \wedge df_8$,

pf_{24}: $df_5 \wedge df_7$, pf_{25}: $df_6 \wedge df_7$, pf_{26}: $df_5 \wedge df_9$, pf_{27}: $df_5 \wedge df_{10}$, pf_{28}: $df_6 \wedge df_9$,

pf_{29}: $df_6 \wedge df_{10}$, pf_{30}: $df_5 \wedge df_{11}$, pf_{31}: $df_6 \wedge df_{11}$, pf_{32}: $df_7 \wedge df_8$ }

Step 23: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features*. So,

Set of Unset Features = { df_4, df_5, df_6, df_7, df_8, df_9, df_{10} }

Set of True Features = { df_2, df_3 }

Set of False Features = { df_1 }

Step 24: *Method User Interface* updates the value of features in Q_3:

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3) [X] *Yes* [ ] *No*
- *Requirements management plan* (df_4) [ ] *Yes* [ ] *No*

Step 25: Using *Method User Interface*, user selects df_4 = '*No*' (df_4 = *FALSE*):

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3) [X] *Yes* [ ] *No*
- *Requirements management plan* (df_4) [ ] *Yes* [X] *No*

Step 26: *Control Logic 1* receives df_4 =*FALSE* and performs simplification of *Map from Features to Variants*. So, *Map from Features to Variants* becomes

Map from Features to Variants = {

pf_1: $FALSE$, pf_2: $TRUE$, pf_3: $TRUE$, pf_4: $FALSE$, pf_5: df_5, pf_6: df_6,

pf_7: df_7, pf_8: df_8, pf_9: df_9, pf_{10}: df_{10}, pf_{11}: df_{11}, pf_{12}: $FALSE$, pf_{13}: $FALSE$, pf_{14}: $TRUE$, pf_{15}: $FALSE$, pf_{16}: df_5, pf_{17}: df_6, pf_{18}: $FALSE$, pf_{19}: $FALSE$, pf_{20}: $df_3 \wedge df_7$, pf_{21}: $FALSE$, pf_{22}: $df_3 \wedge df_8$, pf_{23}: $FALSE$, pf_{24}: $df_5 \wedge df_7$, pf_{25}: $df_6 \wedge df_7$, pf_{26}: $df_5 \wedge df_9$, pf_{27}: $df_5 \wedge df_{10}$, pf_{28}: $df_6 \wedge df_9$, pf_{29}: $df_6 \wedge df_{10}$, pf_{30}: $df_5 \wedge df_{11}$, pf_{31}: $df_6 \wedge df_{11}$, pf_{32}: $df_7 \wedge df_8$ }

Step 27: *Control Logic 1* updates *Set of Unset Features*, *Set of True Features*, and *Set of False Features.* So,

Set of Unset Features = { df_4, df_5, df_6, df_7, df_8, df_9, df_{10} }

Set of True Features = { df_2, df_3 }

Set of False Features = { df_1, df_4 }

Step 28: *Method User Interface* updates the value of features in Q_3:

Q_3: *Which documents will be developed to Plan Scope Management?*

- *Scope management plan* (df_3) [X] *Yes* [ ] *No*
- *Requirements management plan* (df_4) [ ] *Yes* [X] *No*

Step 29: In principle, users would have to continue and answer the remaining questions down to the last one, but in order not to use much space in this paper, this step simulates users choosing to finish the variant selection procedure.

Step 30: *Method Control Logic 1* makes *Set of true process facts* = {pf_2, pf_3, pf_{14}} and sends it to *Method Control Logic 2.*

Step 31: *Method Control Logic 2* set as $TRUE$ the variants in *Set of Variants* that is bound to *process facts* in *Set of true process facts.* Since *Set of true process facts* = {pf_2, pf_3, pf_14}, the system returns

$TRUE$: ({*PC*}, {*atleast1*(*PC*)}),

$TRUE$: ({*SMP*}, {*atleast1*(*SMP*)}),

$TRUE$: ({*PC, SMP*}, {*precedence*(*PC, SMP*), *response*(*PC, SMP*)}),

Step 32: *Method Control Logic 2* joins all variants in *Set of Variants* that are set as $TRUE$. This is *a Syntactically and Semantically Correct SDL Process* to be provided to *Framework* *for User Support at Run Time*. So,

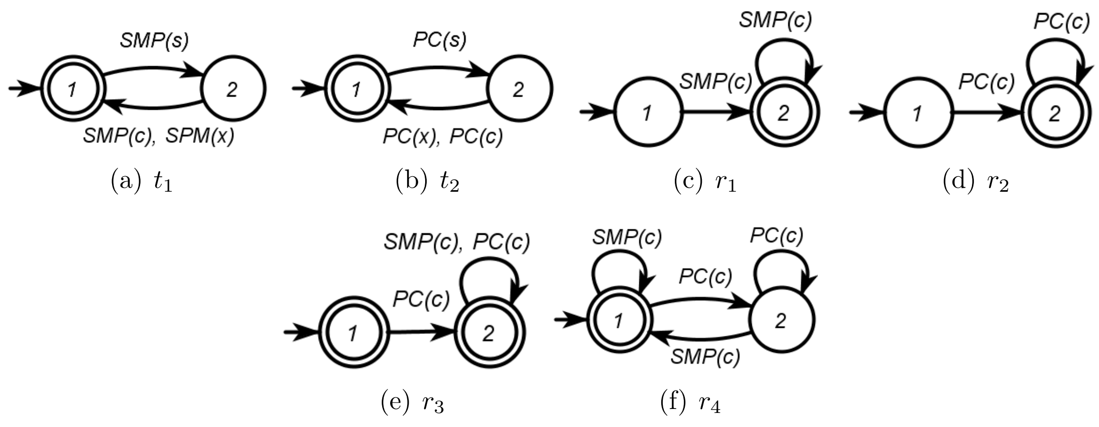
$$\begin{aligned} \textit{SDL Process} = (\ & \{PC,\ SMP\}, \\ & \{\ \textit{atleast1}(PC),\ \textit{atleast1}(SMP), \\ & \ \ \textit{precedence}(PC,\ SMP),\ \textit{response}(PC,\ SMP)\ \}\ ) \end{aligned}$$

Step 33: *Method Control Logic 2* finishes the selection of variants.

6.5.3 At run time

At run time, SVSDL provides users with *Framework for user support at run time*. The sequence of steps with the different interactions among the methods comprising *Framework for user support at run time* is shown next.

Step 1: *Framework for user support at run time* makes the automaton for each task and constraint taken from *Framework for user support at configure time*. These automata are displayed in Figure 61. From these automata, the sequence of markings shown in Figure 62 is created.



(a) t_1 (b) t_2 (c) r_1 (d) r_2 (e) r_3 (f) r_4

Figura 61 – a Syntactically and Semantically SDL process taken from configure time

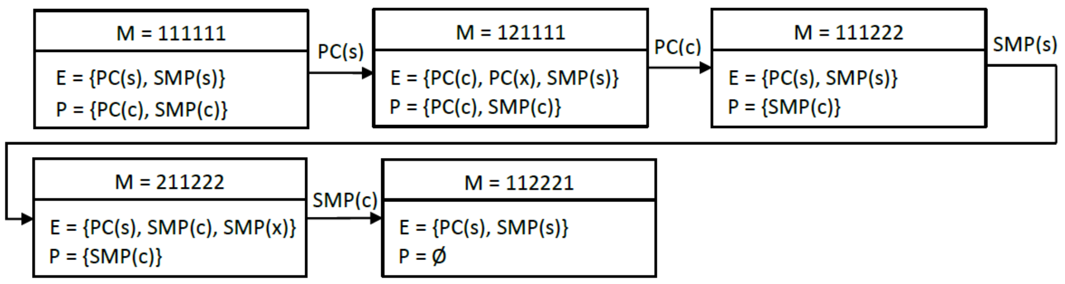


Figura 62 – Sequence of markings

Step 1: *Framework for user support at run time* makes the automaton for each task and constraint.

Step 2: *Framework for user support at run time* calculates marking M = 11111111.

Step 3: *Method Synchronous product* makes *Set of enabled events* = $\{SMP(s), RTM(s), AD(s)\}$.

Step 4: *Method to calculate pending events* makes *Set of pending events* = $\{SMP(c)\}$.

Step 5: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to users.

Step 6: Users choose to execute $SMP(s)$.

Step 7: *Method to update sequence of executed events* updates *Set of executed events* = *Sequence of executed events* = $SMP(s)$.

Step 8: *Method to update current state in automata* takes the last executed event from *Sequence of executed events* to update the current state in each automaton of the SDL process.

Step 9: *Framework for user support at run time* calculates marking M = 21111111.

Step 10: *Method Synchronous product* makes *Set of enabled events* = $\{SMP(c), SMP(x), RTM(s), AD(s)\}$.

Step 11: *Method to calculate pending events* makes *Set of pending events* = $\{SMP(c)\}$.

Step 12: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to users.

Step 13: Users choose to execute $SMP(c)$.

Step 14: *Method to update sequence of executed events* takes $SMP(c)$ and updates *Sequence of executed events* = $SMP(s).SMP(c)$

Step 15: *Method to update current state in automata* takes $SMP(c)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 16: *Framework for user support at run time* calculates marking M = 11122121.

Step 17: *Method Synchronous product* makes *Set of enabled events* = $\{SMP(s), RTM(s), AD(s)\}$.

Step 18: *Method to calculate pending events* makes *Set of pending events* = $\{RTM(c)\}$.

Step 19: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Step 20: Users choose to execute $RTM(s)$.

Step 21: *Method to update sequence of executed events* takes $RTM(s)$ and updates *Sequence of executed events* = $SMP(s).SMP(c).RTM(s)$

Step 22: *Method to update current state in automata* takes $RTM(s)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 23: *Framework for user support at run time* calculates marking M = 12122121.

Step 24: *Method Synchronous product* makes *Set of enabled events* = $\{SMP(s), RTM(c), RTM(x), AD(s)\}$.

Step 25: *Method to calculate pending events* makes *Set of pending events* = $\{RTM(c)\}$.

Step 26: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Step 27: Users choose to execute $RTM(c)$.

Step 28: *Method to update sequence of executed events* takes $RTM(c)$ and updates *Sequence of executed events* = $SMP(s).SMP(c).RTM(s).RTM(c)$

Step 29: *Method to update current state in automata* takes $RTM(c)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 30: *Framework for user support at run time* calculates marking M = 11122212.

Step 31: *Method Synchronous product* makes *Set of enabled events* = $\{SMP(s), RTM(s), AD(s)\}$.

Step 32: *Method to calculate pending events* makes *Set of pending events* = $\{AD(c)\}$.

Step 33: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to user.

Step 34: Users choose to execute $AD(s)$.

Step 35: *Method to update sequence of executed events* takes $AD(s)$ and updates *Sequence of executed events* = $SMP(s).SMP(c).RTM(s).RTM(c).AD(s)$

Step 36: *Method to update current state in automata* takes $AD(s)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 37: *Framework for user support at run time* calculates marking M = 11222212.

Step 38: *Method Synchronous product* makes *Set of enabled events* = $\{SMP(s), RTM(s), AD(c), AD(x)\}$.

Step 39: *Method to calculate pending events* makes *Set of pending events* = $\{AD(c)\}$.

Step 40: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to users.

Step 41: Users choose to execute $AD(c)$.

Step 42: *Method to update sequence of executed events* takes $AD(s)$ and updates *Sequence of executed events* $= SMP(s).SMP(c).RTM(s).RTM(c).AD(s).AD(c)$

Step 43: *Method to update current state in automata* takes $AD(c)$ (the last executed event from *Sequence of executed events*) to update the current state in each automaton of the SDL process.

Step 37: *Framework for user support at run time* calculates marking M = 11122211.

Step 38: *Method Synchronous product* makes *Set of enabled events* $= \{SMP(s), RTM(s), AD(s)\}$.

Step 39: *Method to calculate pending events* makes *Set of pending events* $= \emptyset$.

Step 40: *Method to update sequence of executed events* presents *Set of enabled events* and *Set of pending events* to users.

Since no event is pending, users are able to continue or finish running process, from step 40.

6.6 Conclusion

This paper argued that in recent years, the interest in reference models in project management is increasing and, consequently, it is also increasing in Project Management Information Systems (PMIS). Business processes generated from a reference model are one of the dimensions dealt with in Project Management Information Systems (PMIS). A reference model can generate a set of different business processes for the same application domain.

This paper also quoted studies showing that reusing process models in different contexts can result in a wide range of related process model variants, which belongs to the same process family, and that it is too expensive for companies to design and implement standardized business processes for each actual context in the real world. So, there currently is a high level of interest in gathering common process knowledge to use as reference process models. Thus, an approach to capture and set the variability in a given process model is needed. This approach must be able to represent a family of process variants in a compact, reusable, and maintainable way and should allow process families to be configured so that process variants represent, correctly, the requirements of the respective specific application environments.

This paper selected PMBOK as reference model for project management. The processes to create the application example were selected from PMBOK. The processes in Project Scope Management are modeled using Simple Declarative Language (SDL) (SCHAIDT; SANTOS, 2017b). SDL is a conceptual framework for modeling constraint based processes. Selection of variants is supported by Selection of Variants with Simple Declarative language (SVSDL)

(SCHAIDT; SANTOS, 2017a). SVSDL is a conceptual framework to select variants from processes modeled using SDL.

This paper presented an example of SVSDL application in its three moments: design, configure and run time. At design time, SVSDL provides users with *Framework for user support at design time*. This framework enables users to make *Framework for user support at configure time*. This framework enables users to make *a Syntactically and Semantically Correct SDL Process*. At run time, users are provided with Framework for user support at run time. This framework enables users to run *a Syntactically and Semantically Correct SDL Process*. Thus, these example present the complete SVSDL operation cycle.

The first example uses $Function = Exactly.one$ and questionnaire approach to support the selection of variants. $Function = Exactly.one$ requires that all process facts be mutually exclusive and, consequently, the logical expressions that are bound to these process facts are also mutually exclusive. If $Function = Exactly.one$ then the variants in V_{rpm} cannot be joined to make a variant run. Each variant in V_{sel} is semantically correct. So, if $Function = Exactly.one$ then it is not required that the configurable process model in Example 1 be semantically correct. The previous conditions are complied in Example 1.

The second example uses $Function = Atleast.one$ and questionnaire approach to support the selection of variants. $Function = Atleast.one$ do not require that all the process facts be mutually exclusive and, consequently, the logical expressions that are bound to these process facts are also not mutually exclusive. If $Function = Atleast.one$ then the variants in V_{rpm} can be joined to make a variant that can be run. If $Function = Atleast.one$ then the configurable process model must be semantically correct. The previous conditions are also complied in Example 2.

Each *process fact* must be bound to a logical expression. This logical expression is comprised of logical variables that represent *domain facts*. This condition is fulfilled in Examples 1 and 2. In Examples 1 and 2 there are only the logical rules to bind *process facts* and *domain facts*. There are no other logical rules among domain facts. But, it is possible, if so desired, to define *domain constraints* among domain facts, however this would decrease the number of variants that could be run.

7 Conclusion

Our research's main objective is to propose a variants selection framework from a configurable process model. Configurable process model is modeled by *Simple Declarative Language*. This research sets five *Specific Objectives*. For each *Specific Objective* was defined a set of expected results. Expected results are specific topics to be addressed by each *Specific Objective*. When the five *specific objectives* are fulfilled, the main objective is also fulfilled. Objectives' evaluating is presented next.

7.1 Evaluating compliance with Specific Objectives

Specific Objective 1 was *Define a constraint based language to model the process variants of the framework*. This objective was fulfilled by the Simple Declarative Language (SDL) presented in section 3. SDL provides features to guarantee *Specific Objective 1* is fulfilled. Some of these features are described next. SDL encompasses four constructs: *task*, constraint *atleast*, constraint *precedence*, and constraint *response*. Constraint *atleast* defines that a task must be executed, constraint *precedence* defines the order to execute two tasks, and constraint *response* defines that whenever a task is executed other task must be executed. The fours constructs in SDL are represented by automata. Definition of automata to represent the four constructs in SDL meets $Expected\_Result_{1.1}$ ($ER_{1.1}$). A process modeled by SDL is composed by two sets: a set of tasks and a set of constraints. These sets of tasks and constraints must obey SDL's syntax and semantics. SDL syntax is defined through a set of rules. These rules was an important point to SDL definition. That is because syntax rules support the user to properly specify a process. For example, if the modeler specifies the constraint $precedence(t_i,t_j)$ in set of constraints, then modeler is obligated to specify that the tasks t_i and t_j are in set of tasks. That is very important because constraints refer only to part of tasks events. Constraints do not refer complete tasks. So modelers must to specify tasks in set of tasks. This is an example of SDL syntax rule. SDL syntax rules also support the modeler to specify processes with better performance. For example, modeler cannot specify the constraint $response(t_i,t_i)$. That constraint do not change the process behavior, it only increases the number of states to achieve process objectives. So that constraint is not permitted by the SDL syntax rules. SDL semantics is defined by two rules. These rules was another important point to SDL definition. That is because semantics rules support the user to specify a process to be properly performed. The first semantics rule does not permit set of constraints with constraints $precedence(t_i,t_j)$ and $precedence(t_j,t_i)$ together. Such a constraints combination disable t_i and t_j disabled to be completed. The second semantics rule does not permit set of constraints with constraints *response*(t_i,t_j) and *response*(t_j,t_i) together. Such a constraint combination disable the process to be completed. Definition of syntax and semantics accurate rules for SDL meets $Expected\_Result_{1.2}$ $(ER_{1.2})$.

Specific Objective 2 was *Propose a framework for design time.* This objective was fulfilled by *Framework for user support at design time* presented in Section 4. *Framework for user support at design time* provides a set of methods and data structures to enable the user to make *Framework for user support at run time.* Methods in *Framework for user support at design time* are designed to be performed at a logical sequence. All the data structures made by *Framework for user support at design time* comply with all the requirements at configuration time. *Framework for user support at design time* is, in fact, a specification for nine methods. *Method to make Function*, *Method to make Reference Process Model*, *Method to make Set of Variants*, and *Method to make Map from Features to Variants* enable the modeler to make process variants and mix them into the same constraint based process. These four methods ensure that the SDL syntax and semantics rules are preserved. This is possible due the precise mathematical rules that are obeyed by these methods. These mathematical rules are demonstrated in section 4. These four methods' definitions meet $Expected\_Result_{2.1}$ $(ER_{2.1})$ and $Expected\_Result_{2.2}$ $(ER_{2.2})$. *Method to make Set of features*, *Method to make Set of Domain Constraints*, *Method to define Set of Relations of Precedence to Features*, and *Method to Map Questions to Features* enable the modeler to make *Questionnaire. Questionnaire* enabled the user to select and control all the process features. This is possible due the precise logic rules obeyed by these methods. These logic rules are demonstrated in section 4. Since these four methods enable modeler to make *Questionnaire*, these four methods' definitions meet $Expected\_Result_{2.3}$ $(ER_{2.3})$.

Specific Objective 3 (*SO3*) was *Propose a framework for configuration time.* This objective was fulfilled by *Framework for user support at configuration time* presented in Section 5. *Framework for user support at configuration time* provides a set of methods and data structures to enable the user to make *Framework for user support at run time.* Methods in *Framework for user support at configuration time* are designed to be performed at a logical sequence. All the data structures made by *Framework for user support at configuration time* comply with all the requirements at run time. *Framework for user support at configuration time* is, in fact, a specification for three methods. *Method User Interface* and *Method Logic Control 1* provide support for user to answer the questionnaire. These methods are based on mathematical rules. These rules guarantee that questionnaire is properly answered. Definition of *Method User Interface* and *Method Logic Control 1* meets $Expected\_Result_{3.1}$ $(ER_{3.1})$. *Method Logic Control 1* and *Method Logic Control 2* provides support to simplify and reduce the logic sentences of process features. These two methods are based on logic formalism to support the user to set features to true or false. When the user sets a feature, these methods simplify the logic sentences. These two methods reduce logical variables. That brings a dynamic operation of the questionnaire. Whenever user sets a feature, framework calculates which are the features that user can set next. User cannot set other features before framework make that calculation. Framework also provides methods to select the process variants in accord to syntax and semantics rules, no additional calculation is required to ensure syntactic and semantically correctness. This happens because the logic rules to make each process variant are defined at design time. Definition of *Method Logic Control 1* and *Method Logic Control 2* meets $Expected\_Result_{3.2}$ ($ER_{3.2}$).

Specific Objective 4 (*SO4*) is *Propose a framework for run time.* This objective was fulfilled by *Framework for user support at run time* presented in Section 5. *Framework for user support at run time* provides a set of methods and data structures to enable the user to answer the questionnaire. Methods in *Framework for user support at configuration time* are designed to be performed at a logical sequence. *Framework for user support at run time* provides four methods: *Method to calculate the pendent events*, *Method product synchronous*, *Method to update sequence of executed events*, and *Method to update current state in automata.* These four methods enable the user to run any process modeled by SDL. *Method to calculate the pendent events* supports the user to execute pendent events at each process step. Definition of *Method to calculate the pendent events* meets $Expected\_Result_{4.1}$ ($ER_{4.1}$). *Method product synchronous* supports the user to execute enable events at each process step. Definition of *Method product synchronous* meets $Expected\_Result_{4.2}$ ($ER_{4.2}$). SDL framework offers a great advantageous: no event sequence is calculated at design time, events sequences are calculated only at run time. This happens because SDL framework guarantee syntax and semantics consistency at design time. That is very important in variants selection context because checking syntax and semantics at run time requires very complicated methods. This could bring process performance reduction.

Specific Objective 5 (*SO5*) was *Demonstrate the application of the framework.* This objective was fulfilled by examples presented in Section 6. Section 6 presents reference process models' fundamentals. One of these is Process Management Body Of Knowledge (PMBOK). PMBOK fundamentals' description in Section 6 meets $Expected\_Result_{5.1}$ ($ER_{5.1}$). PMBOK encompasses management, monitoring and control activities. For each new project, these activities are performed in different conditions. Modeling PMBOK processes by imperative languages tends to be hard and confuse. This happens because PMBOK processes tend to be repeatedly executed. For example, while the project management planning is developed it is impossible to define how many times *Scope management plan* will be modified during the project execution. In general, project team define a initial scope, but at most of cases, project requirements need to be modified due several reasons, including factors related to finances, quality, resources, time, among others. As from that argument, article in Section 6 presents two examples to model Project Scope Management processes. Project Scope Management is one PMBOK knowledge area. Article in Section 6 describes five processes: Collect Requirements, Define Scope, Create WBS, Verify Scope and Control Scope. In the two application examples, Project Scope Management processes are modeled by SDL. These processes generate processes variants, i.e. the application contexts. Process variants context (application contexts) meets *Expected_Result*$_{5.2}$ ($ER_{5.2}$). Section 6 also demonstrates examples to model, configure and run process variants from Project Scope Management. These examples meet *Expected_Result*$_{5.3}$ ($ER_{5.3}$). Examples in Section 6 obligate the user to update PMBOK documents whenever some tasks are executed. That condition is precisely one of PMBOK features. In other words, modeling PMBOK by SDL (a constraint based language) eased comply with PMBOK features. Article in Section 7 demonstrated the appropriate operation of the SVSDL methods presented in Section 5.

7.2 Main contribution and its originality

Previous subsection analysed each of the five research's *Specific Objectives.* It was demonstrated that the five research's*Specific Objectives* were fulfilled. Since the five research's *Specific Objectives* were fulfilled, the research's *Main Objective* was fulfilled. Since the research's *Main Objective* was fulfilled, this research also provided its main contribution: *Propose a conceptual framework to select variants as from constraints based processes.* That conceptual framework is *Selection of Variants with Simple Declarative Language* (SVSDL). SVSDL encompasses three frameworks: *Framework for design time*, *Framework for configuration time* and *Framework for run time.* Each of these frameworks encompasses a set of methods. *Framework for design time* methods support user to make the *Framework for configuration time. Framework for configuration time* methods support user to make a constraints based process modeled by *Simple Declarative Language* (SDL). *Framework for run time* methods support user to perform the *SDL process.* At design time, the modeler gathers all relevant data to design *Framework for configuration time. Framework for design time* requires greater work to be performed by the modeler since it is expected to be performed only one time. *Framework for configuration time* will be available to be used repeatedly by users. Whenever *Framework for configuration time* is performed, a new SDL process is performed by *Framework for run time.* With respect to originality, SVSDL provides consistent methods to combine variants selection fundamentals (variability) with constraints based processes fundamentals (looseness). Literature review has demonstrated that there is not work which propose that combination. This is the originality of this research.

7.3 Secondary contributions

The process to achieve the research's *Secondary* and *Main Objectives* brought naturally others secondary contributions. They are described next:

- Development of an approach based on *Supervisory Control Theory* to model constraints based processes. That is *Article 1*.

- Development of an approach to model variation points in pre-specified processes through constraints based language. That is *Article 2*.
- Development of SDL framework to design and run constraints based processes. That is *Article 3*.

7.4 Limitations

Although SVSDL framework complies with research's *Specific* and *Main Objectives*, SVSDL framework presents at least two limitations. These limitations are described in the following:

- First limitation concerns to validation of logic relations. SVSDL presents methods to define logic relations to three levels: *level 1*, *level 2* and *level 3*. They are described in the following. *Level 1*: modeler specifies logic relations just between *Domain Facts*. *Level 2*: modeler specifies logic relations between *Domain Facts* and *Processes Facts*. *Level 3*: modeler specifies logic relations just between *Process Facts*. Although SVSDL provides methods to define logic relations to these levels, it does not offer support to validation if the modeler specifies logic relations to *level 1*. In other words, SVSDL is able to identify when logic relations of *level 2* and *level 3* have some contradiction, but it is not able to identify such contradictions if the modeler also specify logic relations to *level 1*. If modeler also specifies logic relations to *level 1*, then some logic external support is required.
- Second limitation concerns to Simple Declarative Language (SDL). SDL provides a set of only three constraints: *atleast*, *precedence* and *response*. SDL does not provide, for example, constraints to tasks exclusion. That can be very restrictive in cases where is necessary to model a process that requires events exclusion.

7.5 Future works

Future works are related to the limitations of this research. So we enumerated two future works to be done:

- Specify methods to perform logic validation for logic relations at Level 1 (just between *Domain Facts*).
- Specifying, at least, some constraint to exclude tasks in SVSDL.

Referências

AALST, W. M. V. D. Workflow verification: Finding control-flow errors using petri-net-based techniques. In: *Business Process Management*. [S.l.]: Springer, 2000. p. 161–183. Citado na página 74.

AALST, W. M. V. D. Business process management: a comprehensive survey. *ISRN Software Engineering*, Hindawi Publishing Corporation, v. 2013, 2013. Citado 2 vezes nas páginas 15 e 111.

AALST, W. M. V. D. et al. Correctness-preserving configuration of business process models. In: SPRINGER. *International Conference on Fundamental Approaches to Software Engineering*. [S.l.], 2008. p. 46–61. Citado na página 57.

AALST, W. M. V. D.; WESKE, M. to interorganizational workflows. *Seminal Contributions to Information Systems Engineering: 25 Years of CAiSE*, Springer Science & Business Media, p. 289, 2013. Citado na página 111.

AALST, W. M. van D.; PESIC, M.; SCHONENBERG, H. Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development*, Springer, v. 23, n. 2, p. 99–113, 2009. Citado 2 vezes nas páginas 32 e 71.

AALST, W. M. Van der. Verification of workflow nets. In: SPRINGER. *International Conference on Application and Theory of Petri Nets*. [S.l.], 1997. p. 407–426. Citado na página 76.

AALST, W. M. van der. Business process management as the "killer app" for petri nets. *Software & Systems Modeling*, Springer, v. 14, n. 2, p. 685–691, 2015. Citado 3 vezes nas páginas 72, 73 e 75.

AALST, W. M. van der et al. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, Springer, v. 23, n. 3, p. 333–363, 2011. Citado 3 vezes nas páginas 72, 73 e 75.

AALST, W. M. van der; HIRNSCHALL, A.; VERBEEK, H. An alternative way to analyze workflow graphs. In: SPRINGER. *International Conference on Advanced Information Systems Engineering*. [S.l.], 2002. p. 535–552. Citado na página 74.

AALST, W. Van der et al. Conceptual model for online auditing. *Decision Support Systems*, Elsevier, v. 50, n. 3, p. 636–647, 2011. Citado na página 38.

ALLWEYER, T. *BPMN 2.0: introduction to the standard for business process modeling*. [S.l.]: BoD–Books on Demand, 2016. Citado na página 16.

ALOTAIBI, A. B.; MAFIMISEBI, O. P. Project management practice: Redefining theoretical challenges in the 21st century. *Project Management*, v. 7, n. 1, 2016. Citado na página 156.

AREVALO, C. et al. A metamodel to integrate business processes time perspective in bpmn 2.0. *Information and Software Technology*, Elsevier, v. 77, p. 17–33, 2016. Citado na página 15.

ASADI, M. et al. Development and validation of customized process models. *Journal of Systems and Software*, Elsevier, v. 96, p. 73–92, 2014. Citado na página 112.

ASSY, N.; CHAN, N. N.; GAALOUL, W. An automated approach for assisting the design of configurable process models. *IEEE Transactions on Services Computing*, IEEE, v. 8, n. 6, p. 874–888, 2015. Citado 2 vezes nas páginas 113 e 158.

ASSY, N.; GAALOUL, W. Configuration rule mining for variability analysis in configurable process models. In: SPRINGER. *International Conference on Service-Oriented Computing.* [S.l.], 2014. p. 1–15. Citado na página 112.

ASSY, N.; GAALOUL, W. Extracting configuration guidance models from business process repositories. In: SPRINGER. *International Conference on Business Process Management.* [S.l.], 2015. p. 198–206. Citado na página 112.

AYORA, C. et al. Towards run-time flexibility for process families: open issues and research challenges. Springer, 2012. Citado 6 vezes nas páginas 57, 60, 61, 111, 112 e 158.

AYORA, C. et al. Variability management in process families through change patterns. *Information and Software Technology*, Elsevier, v. 74, p. 86–104, 2016. Citado na página 17.

AYORA, C. et al. Dealing with variability in process-aware information systems: language requirements, features, and existing proposals. Universität Ulm, 2013. Citado 2 vezes nas páginas 57 e 112.

AYORA, C. et al. Enhancing modeling and change support for process families through change patterns. In: *Enterprise, Business-Process and Information Systems Modeling.* [S.l.]: Springer, 2013. p. 246–260. Citado 5 vezes nas páginas 17, 56, 111, 114 e 158.

AYORA, C. et al. Vivace: A framework for the systematic evaluation of variability support in process-aware information systems. *Information and Software Technology*, Elsevier, v. 57, p. 248–276, 2015. Citado 13 vezes nas páginas 16, 17, 20, 22, 57, 111, 112, 113, 114, 117, 118, 158 e 159.

BARR, J. et al. Clinical practice guidelines for the management of pain, agitation, and delirium in adult patients in the intensive care unit. *Critical care medicine*, LWW, v. 41, n. 1, p. 263–306, 2013. Citado na página 17.

BENTLEY, C. *The PRINCE2 Practitioner: From Practitioner to Professional.* [S.l.]: Routledge, 2015. Citado na página 157.

BERGER, T. et al. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, IEEE, v. 39, n. 12, p. 1611–1640, 2013. Citado na página 111.

BPMN, B. P. Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*, 2011. Citado na página 61.

BRAGLIA, M.; FROSOLINI, M. An integrated approach to implement project management information systems within the extended enterprise. *International Journal of Project Management*, Elsevier, v. 32, n. 1, p. 18–29, 2014. Citado na página 158.

BRAUN, R. et al. Clinical processes from various angles-amplifying bpmn for integrated hospital management. In: IEEE. *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on.* [S.l.], 2015. p. 837–845. Citado na página 15.

BUIJS, J. C.; DONGEN, B. F. van; AALST, W. M. van der. Mining configurable process models from collections of event logs. In: *Business Process Management.* [S.l.]: Springer, 2013. p. 33–48. Citado na página 114.

CAGLIANO, A. C.; GRIMALDI, S.; RAFELE, C. Choosing project risk management techniques. a theoretical framework. *Journal of Risk Research*, Taylor & Francis, v. 18, n. 2, p. 232–248, 2015. Citado na página 158.

CANIËLS, M. C.; BAKENS, R. J. The effects of project management information systems on decision making in a multi project environment. *International Journal of Project Management*, Elsevier, v. 30, n. 2, p. 162–175, 2012. Citado na página 158.

CARDOSO, A. F. C. Applicability of it service management in the migration to cloud computing. Universidade Portucalense, 2015. Citado na página 17.

CARVALHO, R. M. de et al. Comparing condec to cmmn: Towards a common language for flexible processes. In: IEEE. *Model-Driven Engineering and Software Development (MODELSWARD), 2016 4th International Conference on.* [S.l.], 2016. p. 233–240. Citado na página 16.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to discrete event systems.* [S.l.]: Springer, 2008. Citado na página 34.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to discrete event systems.* [S.l.]: Springer Science & Business Media, 2009. Citado na página 65.

CASTKA, P.; CORBETT, C. J. et al. Management systems standards: Diffusion, impact and governance of iso 9000, iso 14000, and other management standards. *Foundations and Trends (R) in Technology, Information and Operations Management*, now publishers, v. 7, n. 3-4, p. 161–379, 2015. Citado na página 157.

CESTARI, J. M. A. et al. Enhancing flexibility in business process management using declarative languages. In: INSTITUTE OF INDUSTRIAL ENGINEERS-PUBLISHER. *IIE Annual Conference. Proceedings.* [S.l.], 2014. p. 1800. Citado 2 vezes nas páginas 72 e 87.

CHINOSI, M.; TROMBETTA, A. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, Elsevier, v. 34, n. 1, p. 124–134, 2012. Citado na página 74.

CHOU, R. et al. Management of postoperative pain: a clinical practice guideline from the american pain society, the american society of regional anesthesia and pain medicine, and the american society of anesthesiologists' committee on regional anesthesia, executive committee, and administrative council. *The Journal of Pain*, Elsevier, v. 17, n. 2, p. 131–157, 2016. Citado na página 17.

CICCIO, C. D. et al. Generating event logs through the simulation of declare models. In: SPRINGER. *Workshop on Enterprise and Organizational Modeling and Simulation.* [S.l.], 2015. p. 20–36. Citado 2 vezes nas páginas 21 e 71.

CICCIO, C. D.; MARRELLA, A.; RUSSO, A. Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics*, Springer, v. 4, n. 1, p. 29–57, 2015. Citado na página 15.

CLEMPNER, J. An analytical method for well-formed workflow/petri net verification of classical soundness. *International Journal of Applied Mathematics and Computer Science*, v. 24, n. 4, p. 931–939, 2014. Citado na página 74.

CLEMPNER, J. Verifying soundness of business processes: A decision process petri nets approach. *Expert Systems with Applications*, Elsevier, v. 41, n. 11, p. 5030–5040, 2014. Citado na página 74.

COGNINI, R. et al. Using data-object flow relations to derive control flow variants in configurable business processes. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2014. p. 210–221. Citado na página 112.

COHEN, I.; ILUZ, M.; SHTUB, A. A simulation-based approach in support of project management training for systems engineers. *Systems Engineering*, Wiley Online Library, v. 17, n. 1, p. 26–36, 2014. Citado na página 157.

CONFORTO, E. C. et al. Can agile project management be adopted by industries other than software development? *Project Management Journal*, Wiley Online Library, v. 45, n. 3, p. 21–34, 2014. Citado na página 157.

COSTA, M. B.; TAMZALIT, D. Recommendation patterns for business process imperative modeling. In: ACM. *Proceedings of the Symposium on Applied Computing*. [S.l.], 2017. p. 735–742. Citado na página 15.

DEBOIS, S.; HILDEBRANDT, T.; SLAATS, T. Concurrency and asynchrony in declarative workflows. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2015. p. 72–89. Citado na página 72.

DEBOIS, S. et al. The dcr graphs process portal. In: *BPM (Demos)*. [S.l.: s.n.], 2016. p. 7–11. Citado 2 vezes nas páginas 16 e 72.

DERGUECH, W.; BHIRI, S. An automation support for creating configurable process models. In: SPRINGER. *WISE*. [S.l.], 2011. v. 6997, p. 199–212. Citado na página 55.

DERGUECH, W.; VULCU, G.; BHIRI, S. An indexing structure for maintaining configurable process models. *BMMDS/EMMSAD*, Springer, v. 50, p. 157–168, 2010. Citado na página 66.

DIJKMAN, R. M.; ROSA, M. L.; REIJERS, H. A. Managing large collections of business process models-current techniques and challenges. *Computers in Industry*, Elsevier, v. 63, n. 2, p. 91–97, 2012. Citado na página 111.

DIOGO, R. A. et al. A computational control implementation environment for automated manufacturing systems. *International Journal of Production Research*, Taylor & Francis, v. 50, n. 22, p. 6272–6287, 2012. Citado na página 37.

DÖHRING, M.; REIJERS, H. A.; SMIRNOV, S. Configuration vs. adaptation for business process variant maintenance: an empirical study. *Information Systems*, Elsevier, v. 39, p. 108–133, 2014. Citado na página 112.

DUMAS, M.; AALST, W. M. Van der; HOFSTEDE, A. H. T. *Process-aware information systems: bridging people and software through process technology.* [S.l.]: John Wiley & Sons, 2005. Citado na página 15.

DUMAS, M. et al. *Fundamentals of business process management.* [S.l.]: Springer, 2013. v. 1. Citado na página 111.

EL-SAYEGH, S. M. Project risk management practices in the uae construction industry. *International Journal of Project Organisation and Management*, Inderscience Publishers Ltd, v. 6, n. 1-2, p. 121–137, 2014. Citado na página 157.

ESHUIS, R. et al. *Deriving Consistent GSM Schemas from DCR Graphs (Full version).* 2016. Citado na página 72.

ESPARZA, J.; HOFFMANN, P. Reduction rules for colored workflow nets. In: SPRINGER. *International Conference on Fundamental Approaches to Software Engineering.* [S.l.], 2016. p. 342–358. Citado na página 74.

FAHLAND, D. et al. Instantaneous soundness checking of industrial business process models. In: SPRINGER. *International Conference on Business Process Management.* [S.l.], 2009. p. 278–293. Citado na página 74.

FAHLAND, D. et al. Declarative versus imperative process modeling languages: The issue of understandability. In: *Enterprise, Business-Process and Information Systems Modeling.* [S.l.]: Springer, 2009. p. 353–366. Citado na página 32.

FAHLAND, D. et al. Declarative versus imperative process modeling languages: the issue of maintainability. In: SPRINGER. *Business Process Management Workshops.* [S.l.], 2010. p. 477–488. Citado na página 32.

FAQUIH, L. E.; SBAÏ, H.; FREDJ, M. Towards a semantic enrichment of configurable process models. In: IEEE. *Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in.* [S.l.], 2014. p. 1–6. Citado 2 vezes nas páginas 52 e 53.

FAQUIH, L. E.; SBAÏ, H.; FREDJ, M. Configurable process models: A semantic validation. In: IEEE. *Intelligent Systems: Theories and Applications (SITA), 2015 10th International Conference on.* [S.l.], 2015. p. 1–6. Citado 2 vezes nas páginas 52 e 58.

FAVRE, C.; FAHLAND, D.; VÖLZER, H. The relationship between workflow graphs and free-choice workflow nets. *Information Systems*, Elsevier, v. 47, p. 197–219, 2015. Citado na página 72.

FLENDER, C.; FREYTAG, T. Visualizing the soundness of workflow nets. *Algorithms and Tools for Petri Nets (AWPN 2006), University of Hamburg, Germany, Department Informatics Report*, v. 267, p. 47–52, 2006. Citado na página 74.

FURTERER, S. L. *Lean Six Sigma in service: applications and case studies.* [S.l.]: CRC Press, 2016. Citado na página 157.

GERTH, R. et al. Simple on-the-fly automatic verification of linear temporal logic. In: *Protocol Specification, Testing and Verification XV.* [S.l.]: Springer, 1996. p. 3–18. Citado na página 78.

GIACOMO, G. D. et al. Declarative process modeling in bpmn. In: SPRINGER. *International Conference on Advanced Information Systems Engineering*. [S.l.], 2015. p. 84–100. Citado na página 16.

GIRAULT, C.; VALK, R. *Petri nets for systems engineering: a guide to modeling, verification, and applications*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 74.

GOEDERTIER, S.; VANTHIENEN, J.; CARON, F. Declarative business process modelling: principles and modelling languages. *Enterprise Information Systems*, Taylor & Francis, v. 9, n. 2, p. 161–185, 2015. Citado 3 vezes nas páginas 15, 19 e 71.

GOTTSCHALK, F.; AALST, W. M. van der; JANSEN-VULLERS, M. H. Configurable process models—a foundational approach. In: *Reference Modeling*. [S.l.]: Springer, 2007. p. 59–77. Citado 2 vezes nas páginas 54 e 55.

GÖTZFRIED, M. et al. Enabling a sap erp based standard software solution for bpm. In: SPRINGER. *International Conference on Subject-Oriented Business Process Management*. [S.l.], 2013. p. 153–165. Citado na página 112.

GRAMBOW, G.; OBERHAUSER, R.; REICHERT, M. On the fundamentals of intelligent process-aware information systems. In: *Advances in Intelligent Process-Aware Information Systems*. [S.l.]: Springer, 2017. p. 1–13. Citado na página 15.

GRÖNER, G. et al. Modeling and validation of business process families. *Information Systems*, Elsevier, v. 38, n. 5, p. 709–726, 2013. Citado 2 vezes nas páginas 16 e 56.

GRÖNER, G. et al. Validation of families of business processes. In: SPRINGER. *Advanced Information Systems Engineering*. [S.l.], 2011. p. 551–565. Citado na página 56.

GUPTA, S. Workflow and process mining in healthcare. *Master's Thesis, Technische Universiteit Eindhoven*, 2007. Citado na página 53.

HACHICHA, E. et al. A configurable resource allocation for multi-tenant process development in the cloud. In: SPRINGER. *International Conference on Advanced Information Systems Engineering*. [S.l.], 2016. p. 558–574. Citado na página 112.

HAISJACKL, C. et al. Making sense of declarative process models: common strategies and typical pitfalls. In: *Enterprise, Business-Process and Information Systems Modeling*. [S.l.]: Springer, 2013. p. 2–17. Citado na página 71.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Context-based configuration of process variants. 2008. Citado na página 52.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Issues in modeling process variants with provop. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2008. p. 56–67. Citado na página 52.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Managing process variants in the process lifecycle. 2008. Citado na página 52.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Correct configuration of process variants in provop. University of Ulm, Faculty of Electrical Engineering and Computer Science, 2009. Citado na página 57.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Capturing variability in business process models: the provop approach. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 22, n. 6-7, p. 519–546, 2010. Citado na página 52.

HALLERBACH, A.; BAUER, T.; REICHERT, M. Configuration and management of process variants. In: *Handbook on Business Process Management 1*. [S.l.]: Springer, 2010. p. 237–255. Citado na página 52.

HELDMAN, K. *PMP: project management professional exam study guide*. [S.l.]: John Wiley & Sons, 2013. Citado na página 157.

HERZBERG, N.; KIRCHNER, K.; WESKE, M. Modeling and monitoring variability in hospital treatments: A scenario using cmmn. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2014. p. 3–15. Citado na página 112.

HILDEBRANDT, T. et al. Dynamic condition response graphs for trustworthy adaptive case management. In: SPRINGER. *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*. [S.l.], 2013. p. 166–171. Citado 2 vezes nas páginas 72 e 73.

HILDEBRANDT, T.; MUKKAMALA, R. R.; SLAATS, T. Safe distribution of declarative processes. In: *Software Engineering and Formal Methods*. [S.l.]: Springer, 2011. p. 237–252. Citado 2 vezes nas páginas 32 e 33.

HILDEBRANDT, T.; MUKKAMALA, R. R.; SLAATS, T. Declarative modelling and safe distribution of healthcare workflows. In: *Foundations of Health Informatics Engineering and Systems*. [S.l.]: Springer, 2012. p. 39–56. Citado na página 32.

HILDEBRANDT, T. et al. Contracts for cross-organizational workflows as timed dynamic condition response graphs. *The Journal of Logic and Algebraic Programming*, Elsevier, v. 82, n. 5, p. 164–185, 2013. Citado na página 72.

HILDEBRANDT, T. T.; MUKKAMALA, R. R. Declarative event-based workflow as distributed dynamic condition response graphs. *arXiv preprint arXiv:1110.4161*, 2011. Citado 2 vezes nas páginas 72 e 73.

HOFSTEDE, A. T. et al. *Modern Business Process Automation: YAWL and its support environment*. [S.l.]: Springer, 2010. Citado na página 38.

HOMAYOUNFAR, P. Process mining challenges in hospital information systems. In: IEEE. *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*. [S.l.], 2012. p. 1135–1140. Citado 3 vezes nas páginas 52, 53 e 61.

HORNSTEIN, H. A. The integration of project management and organizational change management is now a necessity. *International Journal of Project Management*, Elsevier, v. 33, n. 2, p. 291–298, 2015. Citado na página 157.

HUANG, Y. et al. Ontology-based configuration for service-based business process model. In: IEEE. *Services Computing (SCC), 2013 IEEE International Conference on*. [S.l.], 2013. p. 296–303. Citado na página 111.

IDEN, J.; EIKEBROKK, T. R. Implementing it service management: A systematic literature review. *International Journal of Information Management*, Elsevier, v. 33, n. 3, p. 512–523, 2013. Citado 2 vezes nas páginas 17 e 112.

JIMÉNEZ-RAMÍREZ, A. et al. Generating multi-objective optimized business process enactment plans. In: SPRINGER. *International Conference on Advanced Information Systems Engineering*. [S.l.], 2013. p. 99–115. Citado na página 57.

JIMÉNEZ-RAMÍREZ, A. et al. Generating optimized configurable business process models in scenarios subject to uncertainty. *Information and Software Technology*, Elsevier, v. 57, p. 571–594, 2015. Citado na página 112.

JOSLIN, R.; MÜLLER, R. Relationships between a project management methodology and project success in different project governance contexts. *International Journal of Project Management*, Elsevier, v. 33, n. 6, p. 1377–1392, 2015. Citado na página 157.

JUGDEV, K. et al. An exploratory study of project success with tools, software and methods. *International Journal of Managing Projects in Business*, Emerald Group Publishing Limited, v. 6, n. 3, p. 534–551, 2013. Citado na página 157.

KATZNELSON, L. et al. Acromegaly: an endocrine society clinical practice guideline. *The Journal of Clinical Endocrinology & Metabolism*, Endocrine Society Chevy Chase, MD, v. 99, n. 11, p. 3933–3951, 2014. Citado na página 17.

KAYMAK, U. et al. On process mining in health care. In: IEEE. *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. [S.l.], 2012. p. 1859–1864. Citado na página 53.

KERZNER, H. R. *Project management: a systems approach to planning, scheduling, and controlling*. [S.l.]: John Wiley & Sons, 2013. Citado na página 157.

KESTEN, Y.; PNUELI, A.; RAVIV, L.-o. Algorithmic verification of linear temporal logic specifications. In: SPRINGER. *International Colloquium on Automata, Languages, and Programming*. [S.l.], 1998. p. 1–16. Citado na página 78.

KHERBOUCHE, O. M.; AHMAD, A.; BASSON, H. Using model checking to control the structural errors in bpmn models. In: IEEE. *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*. [S.l.], 2013. p. 1–12. Citado na página 74.

KHLIF, W. et al. A methodology for the semantic and structural restructuring of bpmn models. *Business Process Management Journal*, Emerald Publishing Limited, v. 23, n. 1, p. 16–46, 2017. Citado na página 15.

KOPP, O. et al. The difference between graph-based and block-structured business process modelling languages. *Enterprise Modelling and Information Systems Architectures*, v. 4, n. 1, p. 3–13, 2015. Citado na página 15.

KOSCHMIDER, A.; OBERWEIS, A. How to detect semantic business process model variants? In: ACM. *Proceedings of the 2007 ACM symposium on Applied computing*. [S.l.], 2007. p. 1263–1264. Citado na página 54.

KOUFI, V.; MALAMATENIOU, F.; VASSILACOPOULOS, G. A big data-driven model for the optimization of healthcare processes. In: *MIE*. [S.l.: s.n.], 2015. p. 697–701. Citado na página 16.

KUMAR, A.; YAO, W. Design and management of flexible process variants using templates and rules. *Computers in Industry*, Elsevier, v. 63, n. 2, p. 112–130, 2012. Citado na página 54.

LANDRY, J. P.; MCDANIEL, R. Agile preparation within a traditional project management course. In: *Proceedings of the EDSIG Conference*. [S.l.: s.n.], 2015. p. n3429. Citado na página 157.

LEE, J.; HWANG, S. Variability analysis in process-aware information systems. 2016. Citado 2 vezes nas páginas 15 e 17.

LEITNER, M.; RINDERLE-MA, S. A systematic review on security in process-aware information systems – constitution, challenges, and future directions. *Information and Software Technology*, v. 56, n. 3, p. 273 – 293, 2014. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584913002334>. Citado na página 15.

LENZ, R.; PELEG, M.; REICHERT, M. Healthcare process support: achievements, challenges, current research. *International Journal of Knowledge-Based Organizations (IJKBO)*, v. 2, n. 4, 2012. Citado na página 111.

LENZ, R.; REICHERT, M. It support for healthcare processes–premises, challenges, perspectives. *Data & Knowledge Engineering*, Elsevier, v. 61, n. 1, p. 39–58, 2007. Citado na página 53.

LEON, A. *Enterprise resource planning*. [S.l.]: McGraw-Hill Education, 2014. Citado na página 17.

LI, C.; REICHERT, M.; WOMBACHER, A. Mining business process variants: Challenges, scenarios, algorithms. *Data & Knowledge Engineering*, Elsevier, v. 70, n. 5, p. 409–434, 2011. Citado na página 57.

LIU, G.; JIANG, C. Co-np-hardness of the soundness problem for asymmetric-choice workflow nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, IEEE, v. 45, n. 8, p. 1201–1204, 2015. Citado na página 74.

LIU, G. et al. Complexity of the soundness problem of workflow nets. *Fundamenta Informaticae*, IOS Press, v. 131, n. 1, p. 81–101, 2014. Citado na página 74.

LORENC, A.; SZKODA, M. Customer logistic service in the automotive industry with the use of the sap erp system. In: IEEE. *Advanced Logistics and Transport (ICALT), 2015 4th International Conference on*. [S.l.], 2015. p. 18–23. Citado na página 112.

LU, R. et al. Defining adaptation constraints for business process variants. *BIS*, Springer, v. 9, p. 145â156, 2009. Citado na página 52.

LY, L. T. et al. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, Elsevier, v. 54, p. 209–234, 2015. Citado na página 71.

LYMAN, G. H. et al. Venous thromboembolism prophylaxis and treatment in patients with cancer: American society of clinical oncology clinical practice guideline update 2014. *Journal of Clinical Oncology*, American Society of Clinical Oncology, v. 33, n. 6, p. 654–656, 2015. Citado na página 17.

MAGGI, F. M. Declarative process mining with the declare component of prom. *BPM (Demos)*, v. 32, 2013. Citado 2 vezes nas páginas 16 e 71.

MAGGI, F. M. et al. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: *Business Process Management*. [S.l.]: Springer, 2011. p. 132–147. Citado 4 vezes nas páginas 72, 73, 78 e 79.

MAGGI, F. M.; MOOIJ, A. J.; AALST, W. M. van der. User-guided discovery of declarative process models. In: IEEE. *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*. [S.l.], 2011. p. 192–199. Citado 3 vezes nas páginas 72, 73 e 79.

MAGGI, F. M. et al. Runtime verification of ltl-based declarative process models. In: SPRINGER. *International Conference on Runtime Verification*. [S.l.], 2011. p. 131–146. Citado 3 vezes nas páginas 72, 73 e 79.

MANS, R. et al. Process mining in healthcare: a case study. 2008. Citado na página 53.

MARCELINO-SÁDABA, S. et al. Project risk management methodology for small firms. *International Journal of Project Management*, Elsevier, v. 32, n. 2, p. 327–340, 2014. Citado na página 157.

MARRONE, M. et al. It service management: A cross-national study of itil adoption. *Communications of the association for information systems*, v. 34, 2014. Citado 2 vezes nas páginas 17 e 112.

MARTIN, A. *A combined case-based reasoning and process execution approach for knowledge-intensive work*. Tese (Doutorado), 2016. Citado na página 16.

MARTINSUO, M.; KILLEN, C. P. Value management in project portfolios: Identifying and assessing strategic value. *Project Management Journal*, Wiley Online Library, v. 45, n. 5, p. 56–70, 2014. Citado na página 157.

MECHREZ, I.; REINHARTZ-BERGER, I. Modeling design-time variability in business processes: Existing support and deficiencies. In: *Enterprise, Business-Process and Information Systems Modeling*. [S.l.]: Springer, 2014. p. 378–392. Citado 2 vezes nas páginas 57 e 111.

MENDLING, J.; NEUMANN, G.; NÜTTGENS, M. Yet another event-driven process chain-modelling workflow patterns with yepcs. *Enterprise Modelling and Information Systems Architectures*, v. 1, n. 1, p. 3–13, 2015. Citado na página 74.

MERTENS, S.; GAILLY, F.; POELS, G. Enhancing declarative process models with dmn decision logic. In: SPRINGER. *International Conference on Enterprise, Business-Process and Information Systems Modeling*. [S.l.], 2015. p. 151–165. Citado 2 vezes nas páginas 18 e 71.

MERTENS, S.; GAILLY, F.; POELS, G. Supporting and assisting the execution of flexible healthcare processes. In: ICST (INSTITUTE FOR COMPUTER SCIENCES, SOCIAL-INFORMATICS AND TELECOMMUNICATIONS ENGINEERING). *Proceedings of the 9th International Conference on Pervasive Computing Technologies for Healthcare*. [S.l.], 2015. p. 334–337. Citado 3 vezes nas páginas 18, 19 e 70.

MEYER, A. et al. Modeling and enacting complex data dependencies in business processes. In: *Business Process Management*. [S.l.]: Springer, 2013. p. 171–186. Citado na página 15.

MILANI, F. et al. Modelling families of business process variants: A decomposition driven method. *Information Systems*, Elsevier, v. 56, p. 55–72, 2016. Citado 2 vezes nas páginas 110 e 158.

MINHAS, R. S. *Complexity reduction in discrete event systems*. Tese (Doutorado) — University of Toronto, 2002. Citado na página 36.

MIR, F. A.; PINNINGTON, A. H. Exploring the value of project management: linking project management performance and project success. *International Journal of Project Management*, Elsevier, v. 32, n. 2, p. 202–217, 2014. Citado na página 157.

MONK, E.; WAGNER, B. *Concepts in enterprise resource planning*. [S.l.]: Cengage Learning, 2012. Citado na página 111.

MONTALI, M.; CALVANESE, D. Soundness of data-aware, case-centric processes. *International Journal on Software Tools for Technology Transfer*, Springer, p. 1–24, 2016. Citado 3 vezes nas páginas 72, 73 e 75.

MONTALI, M. et al. Towards data-aware constraints in declare. In: ACM. *Proceedings of the 28th annual ACM symposium on applied computing*. [S.l.], 2013. p. 1391–1396. Citado 3 vezes nas páginas 16, 21 e 71.

MOTAHARI-NEZHAD, H. R.; SWENSON, K. D. Adaptive case management: Overview and research challenges. In: IEEE. *2013 IEEE 15th Conference on Business Informatics*. [S.l.], 2013. p. 264–269. Citado na página 111.

MUKKAMALA, R. R. *A Formal Model For Declarative Workflows*. Tese (Doutorado) — IT University of Copenhagen, 2012. Citado 6 vezes nas páginas 21, 71, 72, 73, 82 e 84.

MUKKAMALA, R. R.; HILDEBRANDT, T.; SLAATS, T. Towards trustworthy adaptive case management with dynamic condition response graphs. In: IEEE. *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*. [S.l.], 2013. p. 127–136. Citado na página 72.

MÜLLER, R.; GLÜCKLER, J.; AUBRY, M. A relational typology of project management offices. *Project Management Journal*, Wiley Online Library, v. 44, n. 1, p. 59–76, 2013. Citado na página 157.

MÜLLER, R. et al. Project management knowledge flows in networks of project managers and project management offices: A case study in the pharmaceutical industry. *Project Management Journal*, Wiley Online Library, v. 44, n. 2, p. 4–19, 2013. Citado na página 157.

MUNDBROD, N.; BEUTER, F.; REICHERT, M. Supporting knowledge-intensive processes through integrated task lifecycle support. In: IEEE. *Enterprise Distributed Object Computing Conference (EDOC), 2015 IEEE 19th International*. [S.l.], 2015. p. 19–28. Citado na página 16.

MUNDBROD, N.; REICHERT, M. Process-aware task management support for knowledge-intensive business processes: findings, challenges, requirements. In: IEEE. *Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International*. [S.l.], 2014. p. 116–125. Citado na página 16.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado na página 74.

MURGUZUR, A. et al. Context-aware staged configuration of process variants@ runtime. In: SPRINGER. *International Conference on Advanced Information Systems Engineering*. [S.l.], 2014. p. 241–255. Citado na página 112.

NATSCHLÄGER, C. et al. Modelling business process variants using graph transformation rules. In: IEEE. *Model-Driven Engineering and Software Development (MODELSWARD), 2016 4th International Conference on*. [S.l.], 2016. p. 65–74. Citado na página 17.

NEVES, S. M. et al. Risk management in software projects through knowledge management techniques: cases in brazilian incubated technology-based firms. *International Journal of Project Management*, Elsevier, v. 32, n. 1, p. 125–138, 2014. Citado na página 157.

OAKLAND, J. S. *Total quality management and operational excellence: text with cases*. [S.l.]: Routledge, 2014. Citado na página 157.

OSIPOVA, E.; ERIKSSON, P. E. Balancing control and flexibility in joint risk management: Lessons learned from two construction projects. *International Journal of Project Management*, Elsevier, v. 31, n. 3, p. 391–399, 2013. Citado na página 157.

PARKER, D. et al. Integration of project-based management and change management: Intervention methodology. *International Journal of Productivity and Performance Management*, Emerald Group Publishing Limited, v. 62, n. 5, p. 534–544, 2013. Citado na página 157.

PATANAKUL, P. et al. Agile project execution. *Project Management ToolBox, Second Edition*, Wiley Online Library, p. 301–322, 2016. Citado na página 157.

PATANAKUL, P.; SHENHAR, A. J. What project strategy really is: The fundamental building block in strategic project management. *Project Management Journal*, Wiley Online Library, v. 43, n. 1, p. 4–20, 2012. Citado na página 157.

PESIC, M. *Constraint-based workflow management systems: shifting control to users*. Tese (Doutorado), 2008. Citado 8 vezes nas páginas 21, 40, 41, 42, 44, 71, 77 e 78.

PESIC, M.; AALST, W. M. van der. A declarative approach for flexible business processes management. In: SPRINGER. *Business Process Management Workshops*. [S.l.], 2006. p. 169–180. Citado na página 32.

PEŠIĆ, M.; BOŠNAČKI, D.; AALST, W. M. van der. Enacting declarative languages using ltl: avoiding errors and improving performance. In: *Model Checking Software*. [S.l.]: Springer, 2010. p. 146–161. Citado 4 vezes nas páginas 71, 72, 73 e 79.

PESIC, M.; SCHONENBERG, H.; AALST, W. M. van der. Declare: Full support for loosely-structured processes. In: IEEE. *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International.* [S.l.], 2007. p. 287–287. Citado 4 vezes nas páginas 32, 72, 79 e 81.

PRITCHARD, C. L.; PMP, P.-R. et al. *Risk management: concepts and guidance.* [S.l.]: CRC Press, 2014. Citado na página 157.

PSOMAS, E.; VOUZAS, F.; KAFETZOPOULOS, D. Quality management benefits through the "soft" and "hard" aspect of tqm in food companies. *The TQM Journal*, Emerald Group Publishing Limited, v. 26, n. 5, p. 431–444, 2014. Citado na página 157.

QUEIROZ, M. H. D.; CURY, J. E. Modular supervisory control of large scale discrete event systems. In: *Discrete Event Systems.* [S.l.]: Springer, 2000. p. 103–110. Citado na página 36.

RAMADGE, P. J.; WONHAM, W. M. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, SIAM, v. 25, n. 1, p. 206–230, 1987. Citado 8 vezes nas páginas 21, 36, 53, 58, 66, 72, 85 e 89.

RAMADGE, P. J.; WONHAM, W. M. The control of discrete event systems. *Proceedings of the IEEE*, IEEE, v. 77, n. 1, p. 81–98, 1989. Citado 9 vezes nas páginas 33, 34, 53, 58, 59, 66, 72, 85 e 89.

RAYMOND, L.; BERGERON, F. Impact of project management information systems on project performance. In: *Handbook on Project Management and Scheduling Vol. 2.* [S.l.]: Springer, 2015. p. 1339–1354. Citado na página 158.

RECKER, J. et al. On the syntax of reference model configuration–transforming the c-epc into lawful epc models. In: SPRINGER. *International Conference on Business Process Management.* [S.l.], 2005. p. 497–511. Citado na página 57.

REES-CALDWELL, K.; PINNINGTON, A. H. National culture differences in project management: Comparing british and arab project managers' perceptions of different planning areas. *International Journal of Project Management*, Elsevier, v. 31, n. 2, p. 212–227, 2013. Citado na página 157.

REICHERT, M.; HALLERBACH, A.; BAUER, T. Lifecycle management of business process variants. In: *Handbook on Business Process Management 1.* [S.l.]: Springer, 2015. p. 251–278. Citado 4 vezes nas páginas 17, 110, 114 e 158.

REICHERT, M. et al. Extending a business process modeling tool with process configuration facilities: The provop demonstrator. 2009. Citado na página 55.

REICHERT, M.; WEBER, B. *Enabling flexibility in process-aware information systems: challenges, methods, technologies.* [S.l.]: Springer Science & Business Media, 2012. Citado 27 vezes nas páginas 8, 15, 16, 17, 22, 32, 38, 39, 40, 41, 42, 43, 48, 51, 52, 53, 58, 60, 61, 62, 110, 111, 112, 116, 117, 118 e 158.

REICHERT, M.; WEBER, B. Process-aware information systems. In: *Enabling Flexibility in Process-Aware Information Systems.* [S.l.]: Springer, 2012. p. 9–42. Citado na página 111.

REIJERS, H. A.; MANS, R.; TOORN, R. A. van der. Improved model management with aggregated business process models. *Data & Knowledge Engineering*, Elsevier, v. 68, n. 2, p. 221–243, 2009. Citado na página 54.

REIJERS, H. A.; SLAATS, T.; STAHL, C. Declarative modeling–an academic dream or the future for bpm? In: *Business Process Management*. [S.l.]: Springer, 2013. p. 307–322. Citado 3 vezes nas páginas 18, 19 e 70.

REINHARTZ-BERGER, I.; SOFFER, P.; STURM, A. A domain engineering approach to specifying and applying reference models. In: *EMISA*. [S.l.: s.n.], 2005. v. 75, p. 50–63. Citado na página 57.

REINHARTZ-BERGER, I.; SOFFER, P.; STURM, A. Organisational reference models: Supporting an adequate design of local business processes. *International Journal of Business Process Integration and Management*, Inderscience Publishers, v. 4, n. 2, p. 134–149, 2009. Citado na página 56.

REISIG, W. *Understanding Petri nets: Modeling techniques, analysis methods, case studies*. [S.l.]: Springer Science & Business Media, 2013. Citado na página 16.

RICKER, L.; LAFORTUNE, S.; GENC, S. Desuma: A tool integrating giddes and umdes. In: IEEE. *Discrete Event Systems, 2006 8th International Workshop on*. [S.l.], 2006. p. 392–393. Citado na página 66.

RIEHLE, D. M. et al. On the de-facto standard of event-driven process chains: How epc is defined in literature. In: *Modellierung*. [S.l.: s.n.], 2016. v. 2, n. 4. Citado na página 21.

ROJAS, E. et al. Process mining in healthcare: A literature review. *Journal of biomedical informatics*, Elsevier, v. 61, p. 224–236, 2016. Citado na página 112.

ROSA, M. L. *Managing variability in process-aware information systems*. Tese (Doutorado) — Queensland University of Technology, 2009. Citado 8 vezes nas páginas 21, 23, 75, 112, 114, 116, 117 e 118.

ROSA, M. L. et al. Business process variability modeling: a survey. *ACM Computing Surveys (CSUR)*, v. 50, n. 1, p. 2, 2017. Citado na página 17.

ROSA, M. L. et al. Questionnaire-based variability modeling for system configuration. *Software and Systems Modeling*, Springer, v. 8, n. 2, p. 251–274, 2009. Citado 2 vezes nas páginas 51 e 54.

ROSA, M. L. et al. Business process variability modeling: A survey. *ACM Computing Surveys*, ACM, v. 50, n. 1, p. 2, 2017. Citado na página 16.

ROSA, M. L.; DUMAS, M.; HOFSTEDE, A. H. ter. Modeling business process variability for design-time configuration. In: *Handbook of Research on Business Process Modeling*. [S.l.]: IGI Global, 2009. p. 204–228. Citado na página 57.

ROSA, M. L. et al. Business process model merging: An approach to business process consolidation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 22, n. 2, p. 11, 2013. Citado 7 vezes nas páginas 17, 57, 110, 112, 113, 114 e 158.

ROSE, K. H. A guide to the project management body of knowledge (pmbok® guide)—fifth edition. *Project Management Journal*, Wiley Online Library, v. 44, n. 3, p. e1–e1, 2013. Citado 7 vezes nas páginas 157, 159, 160, 161, 163, 164 e 165.

ROSING, M. von; SCHERUHN, H.-J.; FALLON, R. L. Information modeling and process modeling. 2015. Citado na página 15.

ROVANI, M. et al. Declarative process mining in healthcare. *Expert Systems with Applications*, Elsevier, v. 42, n. 23, p. 9236–9251, 2015. Citado na página 18.

SALIMIFARD, K.; WRIGHT, M. Petri net-based modelling of workflow systems: An overview. *European journal of operational research*, Elsevier, v. 134, n. 3, p. 664–676, 2001. Citado na página 74.

SÁNCHEZ, M. A. Integrating sustainability issues into project management. *Journal of Cleaner Production*, Elsevier, v. 96, p. 319–330, 2015. Citado na página 157.

ŞANDRU, M.; OLARU, M. Critical path method applied to the multi project management environment. In: ACADEMIC CONFERENCES LIMITED. *ECMLG2013-Proceedings For the 9th European Conference on Management Leadership and Governance: ECMLG 2013*. [S.l.], 2013. p. 440. Citado na página 157.

SANTOS, E. A. et al. Modeling business rules for supervisory control of process-aware information systems. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2011. p. 447–458. Citado 2 vezes nas páginas 72 e 87.

SANTOS, E. A. et al. Modeling business rules for supervisory control of process-aware information systems. In: SPRINGER. *Business Process Management Workshops*. [S.l.], 2012. p. 447–458. Citado na página 38.

SANTOS, E. A. P. et al. Modeling constraint-based processes: A supervisory control theory application. *Comput. Sci. Inf. Syst.*, v. 11, n. 4, p. 1229–1247, 2014. Citado 6 vezes nas páginas 8, 16, 21, 72, 86 e 87.

SANTOS, E. P. et al. Supervisory control service for supporting flexible processes. *Industrial Management & Data Systems*, Emerald Group Publishing Limited, v. 113, n. 7, p. 1007–1024, 2013. Citado na página 38.

SARNO, R. et al. Context sensitive grammar for composing business process model variants. In: IEEE. *Science in Information Technology (ICSITech), 2015 International Conference on*. [S.l.], 2015. p. 53–57. Citado na página 21.

SBAI, H.; FREDJ, M.; KJIRI, L. A process pattern for managing evolution of configurable reference process models. In: IEEE. *Information Science and Technology (CIST), 2012 Colloquium in*. [S.l.], 2012. p. 22–25. Citado na página 52.

SBAI, H.; FREDJ, M.; KJIRI, L. To trace and guide evolution in configurable process models. In: IEEE. *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*. [S.l.], 2013. p. 1–4. Citado na página 58.

SBAI, H.; FREDJ, M.; KJIRI, L. A pattern based methodology for evolution management in business process reuse. *arXiv preprint arXiv:1403.6305*, 2014. Citado na página 58.

SCHAIDT, S. et al. Supervision of constraint-based processes: A declarative perspective. In: SPRINGER. *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*. [S.l.], 2013. p. 134–143. Citado 2 vezes nas páginas 72 e 87.

SCHAIDT, S.; SANTOS, E. A. Selection of variants with simple declarative language (svsdl): a conceptual framework to select variants from constraint based process modeled by simple declarative language. *Pontifical University Catholic of Parana*, PUCPR, v. 1, p. 1–45, 2017. Citado 3 vezes nas páginas 159, 167 e 198.

SCHAIDT, S.; SANTOS, E. A. Simple declarative language (sdl): a conceptual framework to model constraint based processes. *Pontifical University Catholic of Parana*, PUCPR, v. 1, p. 1–35, 2017. Citado 10 vezes nas páginas 119, 123, 125, 131, 144, 149, 154, 155, 159 e 197.

SCHAIDT, S. et al. Dealing with constraint-based processes: Declare and supervisory control theory. In: *Advances in Information Systems and Technologies*. [S.l.]: Springer, 2013. p. 227–236. Citado 4 vezes nas páginas 34, 44, 72 e 87.

SCHONENBERG, H. et al. Towards a taxonomy of process flexibility. In: *CAiSE forum*. [S.l.: s.n.], 2008. v. 344, p. 81–84. Citado na página 57.

SCHUNSELAAR, D. M. et al. Configuring configurable process models made easier: An automated approach. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2014. p. 105–117. Citado 2 vezes nas páginas 111 e 158.

SCHUNSELAAR, D. M. et al. Configurable declare: designing customisable flexible models. In: CITESEER. *20th Int. Conf. on Cooperative Information Systems (CoopIS 2012)*. [S.l.], 2012. Citado 2 vezes nas páginas 113 e 159.

SCHUNSELAAR, D. M. et al. Creating sound and reversible configurable process models using cosenets. *BIS*, Springer, v. 117, p. 24–35, 2012. Citado na página 57.

SCHUNSELAAR, D. M. et al. Petra: A tool for analysing a process family. In: *PNSE@ Petri Nets*. [S.l.: s.n.], 2014. p. 269–288. Citado na página 114.

SCHUNSELAAR, D. M. et al. Yawl in the cloud: Supporting process sharing and variability. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2014. p. 367–379. Citado na página 112.

SCHWALBE, K. *Information technology project management*. [S.l.]: Cengage Learning, 2015. Citado na página 156.

SHARMA, D. K.; RAO, V. et al. Individualization of process model from configurable process model constructed in c-bpmn. In: IEEE. *Computing, Communication & Automation (ICCCA), 2015 International Conference on*. [S.l.], 2015. p. 750–754. Citado na página 112.

SHAUL, L.; TAUBER, D. Critical success factors in enterprise resource planning systems: Review of the last decade. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 4, p. 55, 2013. Citado na página 111.

SILVA, D. B. et al. Dealing with routing in an automated manufacturing cell: a supervisory control theory application. *International Journal of Production Research*, Taylor & Francis, v. 49, n. 16, p. 4979–4998, 2011. Citado na página 37.

SIMON, R. W.; CANACARI, E. G. A practical guide to applying lean tools and management principles to health care improvement projects. *AORN journal*, Elsevier, v. 95, n. 1, p. 85–103, 2012. Citado na página 157.

SISTLA, A. P.; CLARKE, E. M. The complexity of propositional linear temporal logics. *Journal of the ACM (JACM)*, ACM, v. 32, n. 3, p. 733–749, 1985. Citado na página 78.

SLAATS, T. et al. *Flexible process notations for cross-organizational case management systems*. [S.l.]: IT University of Copenhagen, Theoretical computer science section, 2016. Citado na página 15.

SLAATS, T. et al. Exformatics declarative case management workflows as dcr graphs. In: *Business Process Management*. [S.l.]: Springer, 2013. p. 339–354. Citado 2 vezes nas páginas 16 e 71.

SNYDER, C. S. A guide to the project management body of knowledge: Pmbok (®) guide. In: PROJECT MANAGEMENT INSTITUTE. [S.l.], 2014. Citado na página 18.

STROPPI, L. J. R.; CHIOTTI, O.; VILLARREAL, P. D. Defining the resource perspective in the development of processes-aware information systems. *Information and Software Technology*, Elsevier, v. 59, p. 86–108, 2015. Citado na página 15.

SU, R.; WONHAM, W. M. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, Springer, v. 14, n. 1, p. 31–53, 2004. Citado na página 36.

TEALEB, A.; AWAD, A.; GALAL-EDEEN, G. Context-based variant generation of business process models. In: *Enterprise, Business-Process and Information Systems Modeling*. [S.l.]: Springer, 2014. p. 363–377. Citado na página 112.

TEALEB, A.; AWAD, A.; GALAL-EDEEN, G. Towards ram-based variant generation of business process models. In: SPRINGER. *International Conference on Service-Oriented Computing*. [S.l.], 2015. p. 91–102. Citado na página 112.

TELANG, P. R.; KALIA, A. K.; SINGH, M. P. Modeling healthcare processes using commitments: An empirical evaluation. *PloS one*, Public Library of Science, v. 10, n. 11, p. e0141202, 2015. Citado na página 18.

TENERA, A.; PINTO, L. C. A lean six sigma (lss) project management improvement model. *Procedia-Social and Behavioral Sciences*, Elsevier, v. 119, p. 912–920, 2014. Citado na página 157.

TOO, E. G.; WEAVER, P. The management of project management: A conceptual framework for project governance. *International Journal of Project Management*, Elsevier, v. 32, n. 8, p. 1382–1394, 2014. Citado na página 157.

TORRES, V. et al. A qualitative comparison of approaches supporting business process variability. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2012. p. 560–572. Citado 2 vezes nas páginas 57 e 112.

TRUSSON, C. R.; DOHERTY, N. F.; HISLOP, D. Knowledge sharing using it service management tools: conflicting discourses and incompatible practices. *Information Systems Journal*, Wiley Online Library, v. 24, n. 4, p. 347–371, 2014. Citado na página 112.

UNGER, M.; LEOPOLD, H.; MENDLING, J. How much flexibility is good for knowledge intensive business processes: a study of the effects of informal work practices. In: IEEE. *System Sciences (HICSS), 2015 48th Hawaii International Conference on*. [S.l.], 2015. p. 4990–4999. Citado 4 vezes nas páginas 15, 18, 19 e 70.

VALENÇA, G. et al. A systematic mapping study on business process variability. *International Journal of Computer Science & Information Technology*, Academy & Industry Research Collaboration Center (AIRCC), v. 5, n. 1, p. 1, 2013. Citado 2 vezes nas páginas 16 e 57.

VIEIRA, A. D.; CURY, J. E. R.; QUEIROZ, M. H. de. A model for plc implementation of supervisory control of discrete event systems. In: IEEE. *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*. [S.l.], 2006. p. 225–232. Citado na página 37.

VOGELAAR, J. et al. Comparing business processes to determine the feasibility of configurable models: a case study. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2011. p. 50–61. Citado na página 57.

VUKOMANOVIĆ, M.; YOUNG, M.; HUYNINK, S. Ipma icb 4.0—a global standard for project, programme and portfolio management competences. *International Journal of Project Management*, Elsevier, v. 34, n. 8, p. 1703–1705, 2016. Citado na página 157.

WESKE, M. Pesoa: Process family engineering in service oriented applications. *Eröffnungskonferenz Forschungsoffensive" Software Engineering*, 2006. Citado na página 21.

WESTERGAARD, M. Better algorithms for analyzing and enacting declarative workflow languages using ltl. In: S. TOUMANI F., W. K. R.-M. (Ed.). *Business Process Management LNCS*. [S.l.]: Springer, 2011. v. 6896, p. 83–98. Citado na página 33.

WONG, K. C.; WONHAM, W. M. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems*, Springer, v. 6, n. 3, p. 241–273, 1996. Citado na página 36.

WONHAM, W. Design software: Xptct. *Systems Control Group, ECE Dept, University of Toronto, updated July*, v. 1, 2013. Citado na página 66.

WONHAM, W. M. *Design Software: XPTCT*. 2011. Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto. Citado 2 vezes nas páginas 40 e 41.

WONHAM, W. M.; RAMADGE, P. J. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, SIAM, v. 25, n. 3, p. 637–659, 1987. Citado 3 vezes nas páginas 72, 85 e 89.

YAN, Z.; DIJKMAN, R.; GREFEN, P. Business process model repositories–framework and survey. *Information and Software Technology*, Elsevier, v. 54, n. 4, p. 380–395, 2012. Citado na página 112.

YANG, Z.; SEN, G.; PING, J. Research on data center with the active-active mode for sap erp system. *Electric Power Information Technology*, v. 1, p. 040, 2013. Citado na página 112.

YONGSIRIWIT, K.; ASSY, N.; GAALOUL, W. A semantic framework for configurable business process as a service in the cloud. *Journal of Network and Computer Applications*, Elsevier, v. 59, p. 168–184, 2016. Citado 2 vezes nas páginas 113 e 158.

YOUSFI, A.; SAIDI, R.; DEY, A. K. Variability patterns for business processes in bpmn. *Information Systems and e-Business Management*, Springer, v. 14, n. 3, p. 443–467, 2016. Citado 2 vezes nas páginas 15 e 17.

ZHANG, H.; HAN, W.; OUYANG, C. Extending bpmn for configurable process modeling. In: IOS PRESS. *Moving Integrated Product Development to Service Clouds in the Global Economy: Proceedings of the 21st ISPE Inc. International Conference on Concurrent Engineering, September 8–11, 2014*. [S.l.], 2014. v. 1, p. 317. Citado 2 vezes nas páginas 113 e 158.

ZUGAL, S. et al. Investigating expressiveness and understandability of hierarchy in declarative business process models. *Software & Systems Modeling*, Springer, v. 14, n. 3, p. 1081–1103, 2015. Citado na página 15.