UNIVERSITY OF LEICESTER

GOAL COMPLIANCE ASSURANCE FOR DYNAMICALLY ADAPTIVE WORKFLOWS

Thesis submitted for the degree of Doctor of Philosophy
at the University of Leicester

by

BUDOOR AHMAD ALLEHYANI

Department of Informatics

University of Leicester

JUNE 2018

ABSTRACT

Business processes capture the functional requirements of an organisation. Today's businesses operate in a very dynamic and complex environment. Thus, the suitability of automation techniques depends on their ability to rapidly and reliably react to change. To react to change rapidly, an adaptation process for business processes is required. This will also satisfy better quality of services, evidenced through performance and availability. The adaptation process includes a need to support self-monitoring of the business processes, detection of a need for a change, decision making on the right change and execution of the change. The adaptation process must be performed in a reliable and automatic manner with minimal user intervention. One of the techniques that enables automatic adaptation is a policy-driven approach, typically E-C-A policies. Policies can change running business processes' behaviour according to changing requirements by inserting, replacing or deleting functionalities. However, there are no assurances over policies' behaviour in terms of the satisfaction of the original goal which is the space that this thesis fills. The presented work provides an approach to support assurances in the face of automated adaptation and changing requirements. To that end, we use trace refinement and ontologies for ensuring goal compliance during adaptation. We present a goal-compliance framework which incorporates adaptation process through E-C-A policies and goal-compliance constraints for assurance purposes. The framework evaluation targets its performance according to two categories: (1) complexity of both processes and adaptation and (2) execution time including adaptation and verification. The evaluation results show that the framework reliably guarantees the satisfaction of the process goal with minimal user intervention. Moreover, it shows a promising performance in which it is a very important aspect of runtime environment.

DECLARATION

This thesis reports on work undertaken in the Department of Informatics, University of Leicester, supervised by Dr. Stephan Reiff-Marganiec. I hereby declare that the contents of this submission have not previously been published for a degree or diploma at any other university or institute.

All the material submitted is the result of my own research, except where otherwise indicated.

The research work presented in some sections has been previously published, in particular:

- The research challenges and an outline of the contribution, represented in Chapter 1 have been published in (Allehyani and Reiff-Marganiec, 2015)

- The background work in Chapter 2, in particular the KAOS goal modelling, the university admission example used in the case study in Chapter 6 and an overview of the framework have been published in (Allehyani and Reiff-Marganiec, 2016)

- The goal-compliance framework, the definition of the goal-dependency link and the domain- task conformance and their corresponding analysis techniques, represented in Chapter 3 have been published in (Allehyani and Reiff-Marganiec, 2017)

Budoor Allehyani
October 2017

ACKNOWLEDGEMENT

Prophet Mohammad (Peace Be Upon Him) said "*Who does not thank people, does not thank Allah "God*"".

Foremost, I praise Allah for giving me patience and health to face all challenges throughout my PhD journey.

I would like to express my sincere gratitude to my supervisor Dr. Stephan Reiff-Marganiec for his invaluable encouragement and support on both an academic level and a personal level over the past four years. For his patience, expert guidance and help I am sincerely grateful.

I would also like to thank Dr. Fer-Jan de Vries for his expert guidance in CSP and for his help with the administration throughout the course of my PhD study.

I would like to acknowledge my sponsor Umm Al-Qura University and the Saudi Arabian Cultural Bureau in London who gave me the chance to do my PhD study and their constant support.

I would like also to offer special thanks to my parents, without their prayers and encouragement this work would not have been possible. I owe a debt of gratitude to my loving family; husband, daughter and son. I am thankful for their patience, kind support and inspiring me to follow my dreams. My sincere thanks also goes to my uncle Khalid Allehyani for supporting me through my education journey.

To my friends, thank you for listening, sharing experiences, giving advice and making this journey in Leicester a memorable. Special thanks to Nisreen, Maryam, Aisha, Mona and Nawal.

DEDICATION

This thesis is dedicated to my auntie Fouza and my uncle Matir (may peace be upon them) for bringing me up like my own parents and looking after me like their own.

Table of Contents

Table of Figures

Table of Tables

ABBREVIATIONS

AC	Autonomic Computing
BPMN	Business Process Model and Notation
BP	Business Process
BPM@RT	Business Process Models at Runtime
CSP	Communicating Sequential Processes
DGT	Domain Goal Task
E-C-A	Event-Condition(s)-Action(s)
FDR	Failures-Divergence Refinement
GORE-for-BP	Goal-Oriented Requirements Engineering for Business Processes
GTD	Goal-Task Dependency
KAOS	Keep All Objectives Satisfied
LTL	Linear Temporal Logic
LTS	Labeled Transition Systems
M@RT	Models@Run.Time
ProBE	Process Behavior Explorer
RE	Requirements Engineering
TDC	Task-Domain Conformance
TTC	Task-Task Consistency
V&V	Validation and Verification
XML	eXtensible Markup Language

Chapter 1 Introduction

1.1 Automated Adaptation

Workflow management technologies play a major role in modeling and automating business processes (BPs) (Georgakopoulos, Hornick and Sheth, 1995). Each business process can be described in terms of processes where they interact to achieve business goals. Workflows are the technique used to capture the core function (intended behavior) of business processes. One of the main issues of workflows is their rigidity, which means once they are structured, they do not extend to capture unstructured events or exceptions. However, the dynamicity of business environment forces any enterprise to react to process changes based on environmental or contextual requirements. Therefore, a critical challenge for any enterprise is to react to process change at runtime in a safe and correct manner.

The ability for workflows to rapidly and reliably respond to change is still a challenging issue. Today's businesses operate in a very dynamic environment, where change is almost constantly required due to customer demands, legislation and changes to the business' nature (e.g. mergers) as well as the desire to work more efficiently. These changes have implications on how the business operates and hence on the processes describing how the business goals are achieved. Typically making such changes is a matter of redesigning the processes, thus involving business analysts and then updating the software executing the processes. Nevertheless, there are some attempts in the field to realise the change and enable flexibility in business processes.

Automation is the key feature in today's software systems (Vogel-Heuser *et al*, 2014). Autonomic Computing (AC) aims at systems that are capable to deal with complexity and uncertainty with minimal human intervention (Computing, 2006). Autonomic systems combine self-* properties including self-configuring, self-healing, self-optimising and self-protecting (Computing, 2006) and (Parashar and Hariri, 2005). Self-configuring means the system is able to detect change itself and react to it automatically. This is also known as self-adaptive systems in the area of software engineering. Adaptability is considered as a software quality assurance (Jamwal, 2010) and some researchers consider it as a requirement in today's systems due to their complexity and dynamicity (Reichert, Rinderle and Dadam, 2003) and (YongLin and Jun, 2008a).

Adaptation could be achieved manually through human intervention, semi-automatically or automatically. There is an ever increasing demand on automated adaptation due to the fact that workflow systems are complex and dynamic (Zander *et al*, 2016). Automated adaptation has been addressed in current approaches, such as (Gorton *et al*, 2007) and (Burmeister *et al*, 2008a). Such approaches resulted in what is called self-adaptive workflows.

A Self-adaptive system, in the context of software engineering, is defined as a system that "evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible", cited in (Salehie and Tahvildari, 2009). On the one hand, self-adaptive systems combine two promising features: automation and adaptation. On the other hand, quality over automated adaptation is a critical issue because automation without management could be the key to severe consequences. The term management in this context can be defined as software assurances which guarantee correctness and consistency issues. Correctness and consistency are related to syntactical (e.g. inheritance), behavioural (e.g. soundness) and semantic issues (e.g. compliance rules). Most of the state-of-the-art focus on syntactical and behavioural correctness, whereas few studies address semantic correctness.

Zander and Krol claim that "…the full power of automated adaptation can only be set free if we find a way to provide context-sensitive, momentary information about the user and the context to the machine and find a way to automatically interpret it" (Zander *et al*, 2016).

In the context of this research, self-adaptive workflows are business processes represented in Business Process Management and Notation (BPMN) diagrams and they are requirements-aware, self-configuring and self-managing. They are self-adaptive as they can detect 'functional' change through triggering data that are available at runtime and change their behaviour accordingly through Event-Condition-Action (E-C-A policies). In addition, they are context-sensitive as policies check context-based conditions before applying the reconfiguration actions. Policies can change the workflow behaviour by deleting/inserting or replacing functionalities at any position and anytime. The current form of the E-C-A policies have no control over 'what' to delete or insert in terms of whether the 'what' satisfying the original goal and complying to the domain

semantic or not. A policy writer might not be aware of the changing context and this might lead to undesirable behaviour.

Adaptation processes consist of four phases: monitoring, detecting, deciding and acting (Salehie and Tahvildari, 2009). Each phase has different challenges, for example the ability of the software to detect the need for change is a detecting challenge. We focus on the acting phase as we aim to control the acting when policies are ready to react in response to change. There are several aspects have been discussed in the literature about the acting phase of the adaptation process in order to tackle the correctness challenges. We are going to mention the most related aspects to our work including self-managing and runtime verification. E-C-A policies (Gorton, 2011) guarantee syntactic correctness over the reconfigured workflows but cannot guarantee that the functional change at requirement level is consistent with the workflow's original goal. To that end, we desire the capabilities to address consistency issues in an automatic manner to ensure workflow's robustness.

1.2 Runtime Verification: Overview

Runtime verification is defined as "the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property" (Leucker and Schallhart, 2009). However, what Validation and Verification (V&V) strategies can be used for self-adaptive systems, what properties to be checked, where and when to verify are the main challenges in the area of self-adaptive systems (De Lemos *et al*, 2013) and (Salehie and Tahvildari, 2009).

Regarding the correctness issues, self-adaptive workflows must be able to automatically verify the change and make a decision whether to accept it or not based on predefined constraints. This is an important and challenging issue to meet the increasing demands on automation in everyday life. It is argued that self-adaptive systems should have self-testing capabilities in order to ensure their correctness (De Lemos *et al*, 2013a). As self-adaptivity is achieved at runtime, self-testing also must be achieved at runtime for quality reasons: (1) software or service availability is a critical aspect especially in business domains and (2) automatic reasoning might reduce the number of errors considering the issue of growing complexity.

Models@Run.Time (Blair, Bencomo and France, 2009a) is a research community, derived from MDE, that uses abstract models as runtime entities to support various V&V objectives (Szvetits and Zdun, 2016). One of the objectives is ensuring consistency and conformance at runtime. This inspires us to provide runtime assurance at a high-level of abstraction in the face of automation and complexity.

1.3 Motivation and Research Hypothesis

Self-adaptive systems incorporate adaptation properties (i.e. self-* properties), domain characteristics and preferences of stakeholders (Salehie and Tahvildari, 2009). Current research to achieve self-adaptivity in a correct manner has focussed on several correctness aspects that provide quality over adaptation (i.e. qualitative correctness). These aspects which we analyse in section 2.2.2., are (i) syntactical, considering correct structure without the needs to analyse its domain, (ii) behavioural, which guarantee correct behaviour and (iii) semantic aspects, considering domain analysis. Our focus in this thesis is on the semantic aspects, however most of the related approaches either provide manual solution or at design phase of the system lifecycle, in which they are unrealistic and insufficient for self-adaptive systems. Self-adaptive systems change their behaviour automatically and at runtime in which the verification process should be held in an automatic manner and at runtime in order to achieve dependability. Semantic correctness is a wide concept comprising the context and the domain knowledge of a system. In Section 2.2.2, we discussed the related approaches in a general sense in terms of semantic correctness and showed that these approaches do not consider goal compliance in the presence of runtime requirements change. However, some approaches consider goal compliance either at design time for modelling purposes or at runtime for updating system specification when the goal is changed. However, ensuring that the system satisfies its goals in the presence of runtime requirements change remains as a challenge. The hypothesis underlying the research in this thesis is as follows:

> Given the goal specification in an explicit manner and linking its functional goals with the functional requirements represented in the workflow specification, efficient runtime verification is feasible to guarantee goal-compliance in the presence of changes at requirements level. The goal compliance assurance can be achieved by existing verification techniques for complex ad hoc change and for larger workflows in a dependable and scalable manner.

1.4 Research Challenges

The following issues related to workflow adaptation and verification are the main challenges to inspire this work:

C1: Instance ad hoc change

Workflow adaptation can be realized at two different levels: process level and instance level (van der Aalst, Wil MP and Jablonski, 2000). The former addresses the need for flexibility by restructuring the whole process when the change is applicable for all instances and it is known as evolutionary change. In contrast, the change at instance level addresses the need for flexibility by temporally applying change per instance and it is valid until the instance terminates and it is known as ad hoc change. It is challenging to change a process on the fly especially if the change is performed online and at instance level. (Gorton *et al*, 2007) have proposed StPowla approach, which provides a promising solution to automatically adapt instance structure. The key feature of StPowla is the separation of concerns as it separates business logic from its functionality. Another feature is that the policy engine guarantees the syntactic correctness when a policy inserts/deletes a task to/from the process. We build our work on the StPowla approach as we believe that "temporary" instance change is a very important aspect in today's workflows. Consider the following two examples as typical:

Example 1: A business might wish to incentivize orders from a specific customer type by providing a special deal: To attract current accounts from customers with annual salaries exceeding GBP 50K, such customers will be send a hamper basket upon registering. This requires that an additional task to be inserted into the workflow, however, this is not a permanent change and also applies to some users of the process, so it is not sensible to redesign the process.

Example 2: An enterprise issues a new rule that all business travel has to be second class and that at least three quotes need to be obtained to prove value for money. This sort of change requires adaptation to all tasks in all processes where travel booking is involved. This type of adaptation, process evolution, is complex as it requires to redesign the whole process. It is also risky as it might miss points that needs adaptation.

C2: Automatic runtime adaptation

Typically, adaptation is achieved through redesign, essentially going back to the drawing board. Slightly more dynamic current approaches to workflow adaptation manually modify running instances in response to change, i.e. authorized users are responsible to make changes to a workflow. However, those approaches do not provide a satisfactory solution since most business processes are complex and dynamic. Hence, instance change at runtime should be achieved online to guarantee service availability.

C3: Compliance and correctness issues

One of the main challenges arising from dynamic workflow adaptation is to ensure that the change does not cause any inconsistences or runtime errors thus guaranteeing that the adapted workflow still satisfies the domain properties within some sensible range of expectations. The goal model is considered as the main reference for workflow in its entire lifetime through and after its development. Therefore, any changes or updates applied to a workflow must be compliant to the original goal, otherwise the workflow might behave in an unexpected way. The most challenging issue related to quality or reliability over adaptation is the semantic issues, in particular the domain or context correctness.

C4: Automatic runtime verification

Self-adaptive system adapts its structure, without human intervention in the midst of its execution. Therefore, the traditional validation and verification (V&V) mechanisms should be tolerated or integrated with self-adaptive systems as they are designed to work at design time where everything is planned. Furthermore, the need to find new V&V techniques to ensure correctness at runtime is being investigated in recent years. Goal satisfaction in self-adaptive systems is defined as one of the major challenges facing the field (Tamura *et al*, 2013). Runtime V&V techniques are characterized by properties including: sensitivity, isolation, incrementality and composability (De Lemos *et al*, 2013). In such manner, the self-adaptive V&V techniques are should be autonomic, intelligent and may be monitored. The balance between adaptation and correctness is of upmost importance and it is a challenging issue to be addressed with minimal user intervention.

1.5 Research Scope and Assumptions

This thesis addresses goal compliance in self-adaptive systems with automatic requirements changing at business level. We consider ad hoc change at requirements level for workflow systems, focusing on control or flow structure. While the dataflow and exception handling are outside the scope of this thesis. The workflows are considered to be isolated (i.e. not complex workflows). The assumptions for this research are as follows:

1. We assume workflows are structured and expressed in BPMN diagrams. Scientific domains have used workflows to structure and execute processes. While in this work the focus is on business processes, we believe that it has merit in the scientific workflow domain, too.
2. We assume goal and BPMN models are correctly defined at design phase and the BPMN is consistent to its goal before adaptation.
3. We have followed the assumption, in the software engineering domain, which states that domain knowledge can be understood and derived from the goal.
4. we assume that in some cases a change cannot be made, in which case the original process will continue to execute (likely resulting in some form of undesired output).

1.6 Research Objectives

The primary aim of this research is to provide assurances that self-adaptive workflows are compliant to their original goals. Self-adaptive workflows change their behaviour according to changing requirements, therefore, the emerging requirements should not deviate from the original goal, see Figure 1-1. In order to achieve that we identify the following objectives:

1. To study how the policy actions (delete/insert) when adapting the requirements model (i.e., BPMN) on the satisfaction of the original goal. In other words, how inserting or deleting new requirements would deviate from the original goal.
2. To define a management link between the requirements model and the goal model in order to be able to measure goal satisfaction in the presence of requirements change at requirements level.
3. To define a semantic-based ontology that captures the requirements in the domain in which we are interested and define the relationship that links the domain knowledge (i.e., requirements), for verification purposes.

4. To develop verification algorithms using existing techniques for runtime assurance over the adaptation processes. The verification algorithms ensure goal-compliance.
5. To evaluate the dependability and applicability of the verification algorithms and check their satisfaction according to the runtime desirable criteria.

1.7 Research Questions

Table 1-1: Research questions linked to research challenges

No.	Research Questions	Related Challenge(s)
RQ1.	How to address goal compliance in the face of automated adaptation and changing requirements? To what extent the changing requirements can be reliably verified against unchanged goal?	C3 and C4
RQ2.	How to decide the satisfaction link among the goal specification and its corresponding requirements model? In other words, how to link the high-level goals to lower-level system components to perform automatic verification at business level? Is understanding the domain from the goal model sufficient to control goal compliance?	C2 and C4
RQ3.	To what extent goal satisfaction can be guaranteed without human intervention? how to balance adaptation and goal satisfaction with minimal user intervention?	C3 and C4
RQ4.	Do the existing verification techniques enable correctness assurance in self-adaptive systems in a realistic and reliable manner?	C3 and C4

This research is motivated by a number of questions arising from the challenges (numbered above C1-C4). Table 1-1 shows the linkage between research challenges and questions. Challenge C1 is not the focus of this research, it explains why we consider the ad hoc adaptation rather than the process evolution.

1.8 Thesis Contribution

This research is motivated by the increasing demand on automation, abstraction and quality over self-adaptivity. However, automation without management would be the key reason for serious consequences. We aim to manage the automated adaptation of workflow systems without sacrificing their functionality. Nevertheless, self-adaptive systems in addition to their ability to detect and react to changes should be able to take the desired actions by preserving correctness criteria. Thus, the automated adaptation of workflow systems must be enhanced with constraints, which guarantee the satisfaction of their original goals.

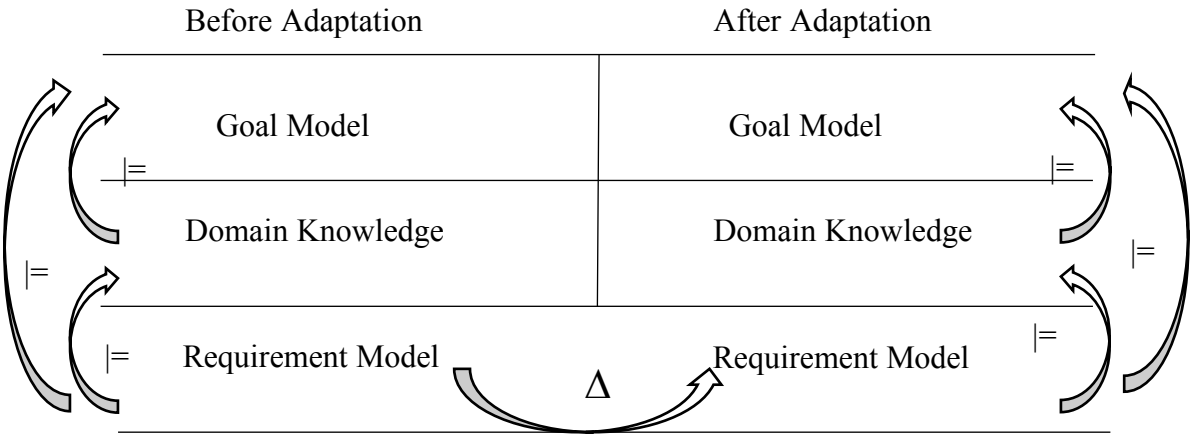


Figure 1-1: Goal compliance in self-adaptive systems

This thesis argues that there is a real need to ensure domain-semantical correctness in self-adaptive workflows beside syntactical as well as generic behavioural properties. The verification process towards that should be automatic and at runtime to meet runtime requirements and automation nature of self-adaptive workflows. Furthermore, automation over assurance process could lead to less effort, error and cost. The main contribution and outcomes of this research are as follows:

1. Goal-compliance algorithms for ensuring the compliant of the new emerging requirements to the original goal.

2. A methodology for linking the goal specification with the workflow specification at a high-level of abstraction for providing assurances on goal satisfaction during workflow adaptation.
3. An ontology-based methodology for providing assurances on goal satisfaction during workflow adaptation.
4. A goal-compliance framework for dynamic adaptation and verification, see Figure 1-2.
5. A proof-of-concept implementation.
6. An evaluation of the goal-compliance framework against the runtime desirable features.

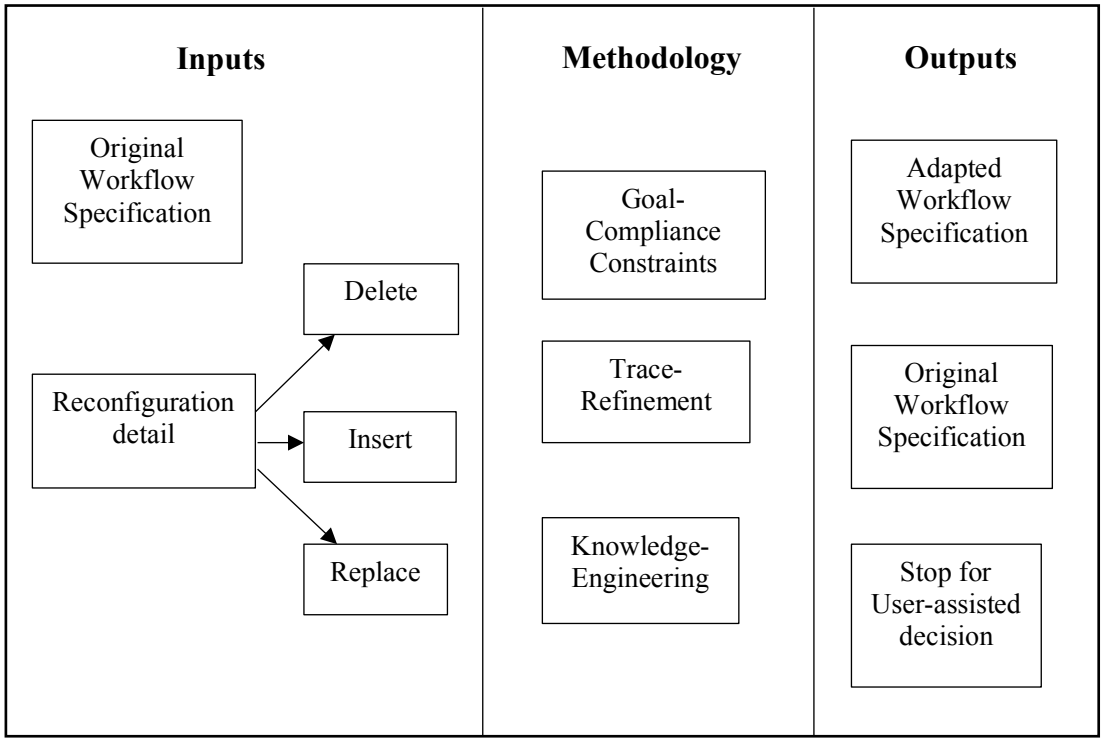


Figure 1-2: A Framework for runtime verification in self-adaptive workflows

The key finding of this research is that the automated requirements adaptation in self-adaptive workflows can be managed to ensure goal and domain correctness. The correctness assurance can be achieved at business level, on running workflows with minimal user intervention and with a promising performance. The challenge in the context of this research is how to balance between the process of achieving adaptation and the process of achieving a quality over adaptation without user intervention.

THESIS STATEMENT

Self-adaptive workflows are able to detect change and take the required action(s) towards change. One of the main challenges to manage self-adaptive workflows is to provide assurances on goal satisfaction. Therefore, we developed an automatic checking mechanisms that can be easily applied to ensure change compliance to the workflow goals during adaptation.

1.9 Thesis Structure

The thesis consists of seven chapters as shown in Figure 1-3. The remaining chapters are described as follows:

Chapter 2 introduces the main background concepts and tools used in this research. It also discusses the state of the art and sheds light on the challenging issues.

Chapter 3 presents the proposed framework, discusses the motivation behind it and its implementation.

Chapter 4 discusses the proposed mechanisms towards runtime V&V of self-adaptive workflows. It illustrates the mechanism and the technique that facilitated goal-compliance check. In particular, it discusses the effect of deleting a task from the process on achieving the goal.

Chapter 5 focuses on studying the effect of inserting a new task to the process on goal achievement. It presents the proposed technique to check the conformance of the new task with the domain.

Chapter 6 shows the feasibility and applicability of our approach by providing evaluation on its performance and accuracy.

Chapter 7 concludes the thesis with a discussion on the capabilities and limitations of our approach highlighting the key findings. It also provides some recommendations and further research.

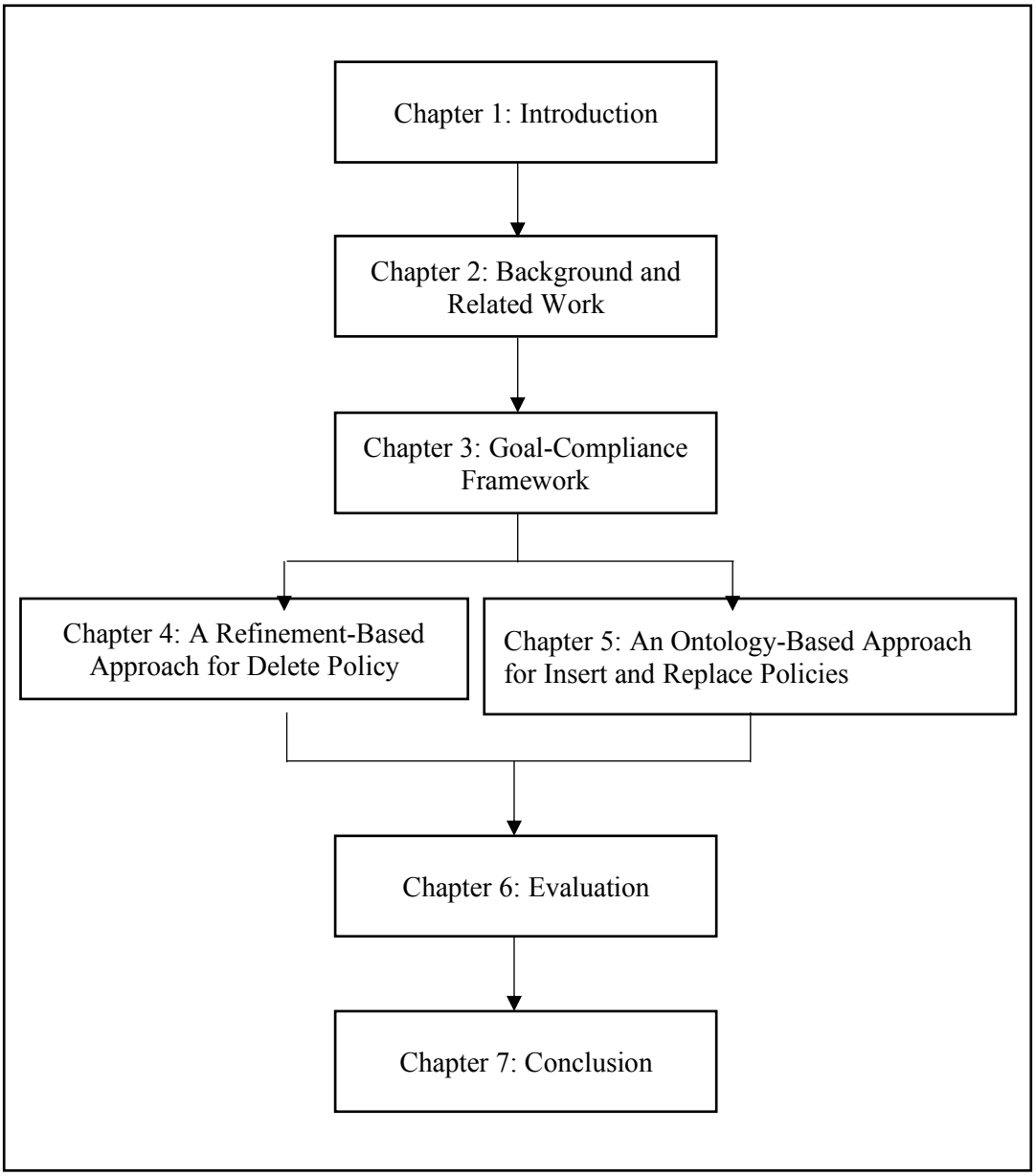


Figure 1-3: Thesis outline

Chapter 2 Background and Related Work

This research is built upon existing approaches and techniques for workflow modelling, reconfiguration and verification. We use and integrated existing solutions in the aforementioned areas to produce a new solution for providing semantical assurance in self-adaptive workflows. In this chapter, we present the mapping from BPMN into Communicating Sequential Processes (CSP), which we use as a modelling and verification language for BPMN, and the E-C-A policies, which we adopt as workflow reconfiguration method. In addition to CSP, we use ontologies for modelling and verification purposes. Furthermore, we discuss related work based on two major concerns: (1) the need for adaptation and (2) the need for assurances to guarantee qualitative adaptation and show the importance of goal assurance in the presence of uncertainty.

2.1 A Running Example: UQU Enrolment System

Our running example considers the specification and verification of Umm Al-Qura University (UQU) enrolment system from the university point of view (*UQU, 2014*) The university admission core process is depicted in Figure 2-1.

At first step, applicants register their personal and educational information and may need to submit relative documents. All applicants must pass GAT and attainment tests before the admission process starts. Applicants who intend to apply for English language program or Medicine must do English test to be able to assign them to the right group. The university then obtains the applicants' test results automatically and checks the requirements specified in a checklist to decide whether the applicant is able to register with the university or not. Check list includes the following conditions:

- The applicant must be Saudi
- The applicant must be medically fit
- High school certificates must be within 3 years
- General Attitude Test (GAT) and attainment test must be within five years
- English proficiency test must be within three years.

After that, the university automatically calculates the ratio for each applicant in order to restrict his/her preferences in advance to the submission process. Note that this restriction is controlled by some conditions and they might differ according to the program itself. For example, some programs require that the nominated applicants must pass their interviews.

The second stage of the enrolment process starts when the university opens the submission system. The applicants should be aware of the start and end date of the submission process. They are then able to apply for programs by preferences. There are different requirements for each program and the university is aware of that in advance to restrict applicant's selections.

The next stage of the enrolment is university nomination as it nominates automatically to the best programs if the conditions are met. Then applicants are informed by emails with the nomination results. They then have the opportunity whether to confirm or withdraw their places.

For those applicants whom accept the nomination result, the university will check their status if they are not enrolled with another university at the same year. If this is satisfied, then applicants will receive their approvals including their university IDs. If not, the university must send them an email to notify them of rejection.

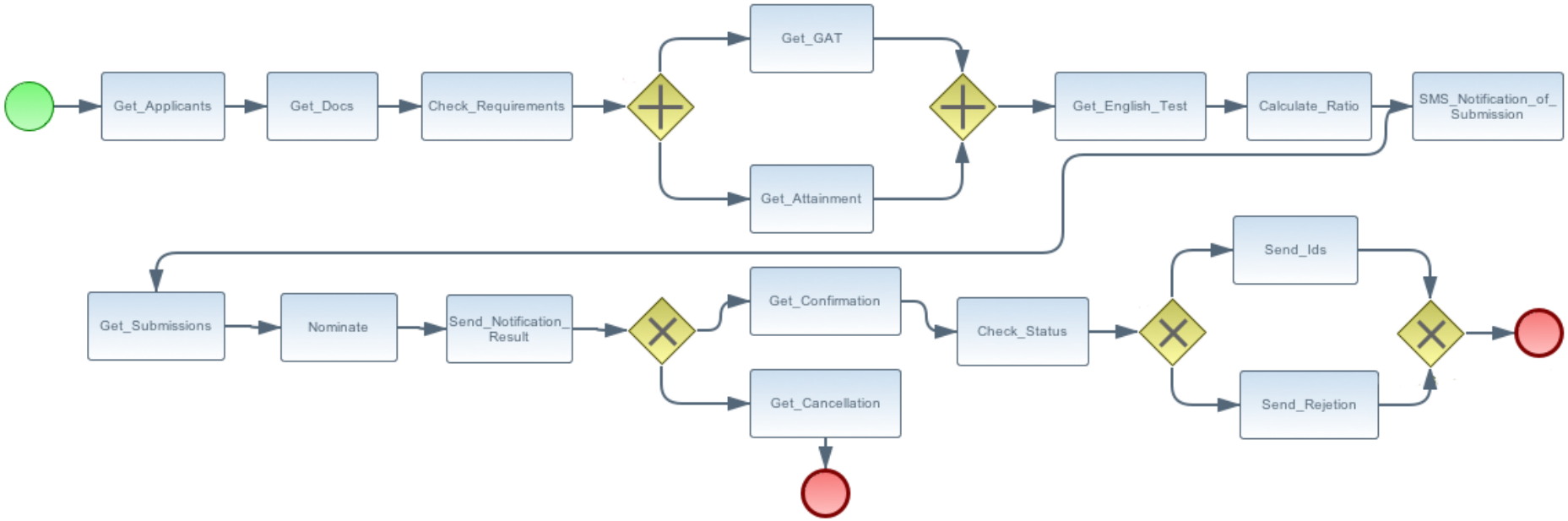


Figure 2-1: Admission process for UQU (adapted from (UQU, 2014), translated from Arabic Scenario)

The university admission process could be highly vulnerable to ad hoc change at runtime, because it is a complex process which needs a collaboration among several parties (e.g. applicants, university admission and ministry of education and other institutions). Modelling all requirements at design time is implacable and might lead to complex and pervasive process. Hence, it is impossible to capture every requirement at design time and this illustrates the need for online and automatic ad hoc change. Furthermore, we show how to automatically verify the ad hoc change in order to ensure goal satisfaction at runtime. The core process captures the functional requirements of an organisation and it is built to logically satisfy its goal.

We are going to refer to this example throughout the thesis to illustrate our approach considering some expected policies that are applicable for the process.

2.2 Background

This section gives an overview of the languages and technologies that we used in this research. Specifically, it illustrates the syntax and semantic of BPMN, KAOS and CSP in order to make it easier to understand the notation used throughout this thesis. It also discusses briefly the mapping of BPMN to CSP and the adaptation policies. Finally, it discusses the ontology and how they are used as a mean of verification.

2.2.1 BPMN Processes and Goals

Business processes are used to describe how a business achieves a goal and also to automate processes by executing them on workflow engines, as discussed in section 3.2. Each business process is described in terms of a workflow, essentially a set of tasks that are conducted in a specific order and the interaction of those tasks with the environment, ultimately capturing how a business goal is achieved. BPMN (*OMG, 2013*) is the de-facto standard in business process modelling. A BPMN diagram is constructed from a set of BPMN elements; activities and their flows. However, BPMN diagram can be easily designed and simulated by BPMN modelling and simulating tools. We use the Eclipse BPMN2 Modeler (*BPMN2Modeler, 2017*) for BPMN modelling. An activity is defined by the BPMN specification as “a generic term for work that company performs in a process” (*OMG, 2013*). It can be atomic task or composite task (i.e. subprocess). A process model represents the requirement model which is constructed at design time. However, it is impossible and impractical to anticipate all activities prior to runtime due to the fact that the business environment is dynamic and complex. However, process

models are prone to change and should be able to react to change efficiently.

Although BPMN is widely accepted for business process modelling, it is insufficient for analysis and verification purposes. This is due to the fact that BPMN lacks a formal semantic such as mathematical-based specification. Business process goals are achieved through logically related tasks in the process model. Furthermore, one of the shortcomings of BPMN is the lack of explicit/formal specification of the goal that it is designed for as well as the lack in mechanisms to measure goal achievement in standard BPMN modelling and simulating tools. Thus, it is difficult to trace and reason about changing BPMN behaviour at runtime. However, there are some approaches which map BPMN semantics to several formal languages such as Petri net (Peterson, 1981) and CSP (Roscoe, 1998) to enable reasoning about BPMN behaviour as theses languages have a formal semantic and are tools supported.

The work by Dijkman (Dijkman *et al*, 2007) has provided a transformation of the BPMN into Petri net semantics throughout a prototypical tool that produce the transformation into a script expressed as Petri Net Markup Language (PNML). PNML is an XML-based format for Petri nets (Weber and Kindler, 2003). The resulting PNML scripts are supported by tools for analyzing the behavioural properties such as soundness.

BPMN is also mapped to CSP semantics (Wong, 2011), as will be explained later in section 2.3.2.1. Similar to Petri net transformation, CSP is supported with tools for behavioural analysis. However, we choose the CSP approach for our verification purposes due to the fact that CSP is characterized by the notion of refinement. In particular, the trace refinement for analysing the trace of the model against a set of desirable properties. In addition, the CSP hiding process helps to form a simple property specification and avoid complex specification as discussed in section 4.6.1.

In addition to the formal mapping of BPMN, there are also attempts to provide explicit linkage among goal specification and the processes they are designed for. Among others we follow Anton's definition of the goal (Antón, McCracken and Potts, 1994) which states that the goal is "high-level objectives of the business, organization or system; they capture the reasons why a system is needed and guide decisions at various levels within the enterprise". In Requirements Engineering (RE), goals are of two types: (i) hard or functional goals capturing the functional aspects of an organization and usually they have

strict criteria to measure satisfaction; (ii) soft or non-functional goals describing the quality aspects the enterprise wishes to achieve and unlike for hard goals there is no explicit criteria for satisfaction. Functional goals describe the strategic goals of an enterprise and each strategic goal is refined by a number of sub-goals or objectives. The strategic goals are qualitative whereas sub-goals are quantitative and usually described as operational objectives that can be measured and assigned to agents.

Goal modelling has been applied as a foundation for business process modelling and redesign. One of the methods applicable here is requirement specification which relates business goals to functional and non-functional system components. In this research, we used Keep All Objectives Satisfied (KAOS) (Van Lamsweerde, 1991) for goal modelling as it provides formal declaration of goals in terms of functional objectives, which are easily mapped to lower level system components (BPMN tasks). KAOS is a "KAOS is a methodology for requirements engineering enabling analysts to build requirements models and to derive requirements documents from KAOS models" (Respect, 2007). The hierarchy of the goal model in KAOS consists of the strategic goal as a root node, which is then refined to objectives representing the functional requirements. The refinement relation between objectives are of two types; OR and AND. The former indicates that at least one of the objectives must be satisfied, while the latter indicates that all objectives must be satisfied. Goal modelling in KAOS is achieved in two different ways: (1) semi-formal goal structuring model and (2) formal definition in linear temporal logic. The former is supported by a tool called Objectiver (*Objectiver, 2015*) and it helps to graphically depict the goal specification. The formal declaration of goals captures goals in Linear Temporal Logic (LTL) specification with variant patterns as follows:

1. Achieve goals; the target must eventually occur (desire achievement)
2. Cease goals; there must be a state in the future where the target does not occur (disallow achievement)
3. Maintain goals; the target must hold at all time in the future
4. Avoid goals; the target must not hold at all time in the future.

In the context of our work, we just used the first and third patterns as will be seen in Chapter 4.

The goal considers only the logical properties of the process, i.e. the occurrence of certain tasks, but does not consider the temporal properties in terms of the order among tasks.

However, business rules or compliance rules capture the temporal and may be other desired properties that specify how the requirement model must behave. As our focus is on goal satisfaction, we focus on the occurrence properties and neglect the temporal aspects. However, the temporal correctness has been addressed in the literature under the umbrella of compliance rules.

There are two approaches for linking goal model with its process model: top-down and bottom up. Top-down approach aims at designing an explicit goal model and then generating its process model by extracting tasks from objectives while relationship among tasks is inferred from the refinement relationship among objectives at goal level. Bottom-up approach aims at generating a goal model from previously designed process model by converting every task to objectives while relationship among objectives is consistent with the relationship among tasks at process level. In the context of this work, we follow the first approach, Top-down, as we argue that the goal should be explicitly defined to support requirements engineering activities.

2.2.2 CSP

CSP is a formalism to describe communicating processes. A process is defined by a set of events describing the ways it interacts with its environment. These sets of events are known as the process alphabet and are written as Σ. The events are combined using operators and processes advance by acting and reacting to their environment (which is captured by another process). The syntax of CSP is shown in Table 2-2. CSP is a formal mathematical notation and supported by a number of analysis and reasoning tools. Two key reasons for using CSP in our work are: (a) the existence of previous work which transforms BPMN to CSP semantic (Wong, 2011) and (b) the fact that CSP supports the notion of refinement which helps in reasoning about BPMN behaviour. Refinement is an ideal notion for our aim; the implementation of a business process is a refinement of its specification; that it exhibits the correct behavior but contains details that are required for running it.

Table 2-1: CSP Syntax

CSP Operators	Semantics
a → P	Prefix
P ||| P	Parallel interleaving
P |[A | B]| Q	Parallel composition of P and Q sharing events in A∩B
P |[A]| Q	Parallel composition of P and Q sharing events in A
P \ A	Hiding all events in A from P
P □ Q	External choice (deterministic)
P ∏ Q	Internal choice (nondeterministic)
P; Q	Sequential composition
P △ Q	Interruption
Skip	Successful termination
Stop	Deadlock

CSP semantics are either denotational or operational. The denotational semantic is described by three models which are the trace model, failure model and failure-divergence model. They are described briefly below:

1. **Traces refinement:** observes the process behavior in terms of indicating all events a process can engage (safety property)
2. **Failure refinement:** observes the process behavior as in the traces refinement and additionally considers events a process can refuse to engage.
3. **Failure-Divergence refinement:** indicates livelock processes (when a process performs an infinite internal loop).

The second form of semantic that is supported in CSP is the operational semantic and it graphically represents the CSP processes in terms of Labeled Transition Systems (LTS). CSP is supported by tools, specifically The Failures-Divergence Refinement (FDR) and Process Behavior Explorer (ProBE) (*Tools for CSP*) . The FDR is the refinement checker for CSP processes. It checks the refinement relation of a CSP specification and its implementation. It is also used to reason about model properties such as deadlock-freedom and safety by using assertions. ProBE is the CSP animator or simulator. It enables the user to "browse" a CSP process by following events that lead from one state of the process to another. However, we did not use ProBE as the latest version of FDR

enables this feature using the ':graph' command. In the context of this research, this feature helps in visually comparing the trace of the desired behaviour as specified with properties and the actual behaviour or the process implementation.

The notion of CSP refinement leverages the runtime verification of self-adaptive workflows as it enables to compare the trace of the desirable specification (for example a goal specification captured in properties) and the implementation (for example the adapted workflow). In addition, CSP is a modelling as well as verification language. Hence, the property specification can be written directly in CSP, without the need to use different notations from CSP (e.g. LTL) and its satisfaction is calculated directly through the FDR.

2.2.2.1 BPMN to CSP

As discussed above, BPMN can be transformed into different formal languages in order to facilitate analysis and verification. BPMN to CSP is one of those mappings and it is conducted by (Wong, 2011). The transformation can be automatically achieved by Wong's prototype tool. This tool is implemented in Haskell and it takes the XML-based BPMN file as an input and returns its corresponding CSP_m file, where CSP_m is the machine readable CSP for FDR tool. In the following, we are going to give a brief overview of the logic behind this transformation with an example as illustration.

Consider the university admission business process expressed in BPMN as a running example, see Figure 2-1. The process consists of sixteen tasks, five gateways, one abort event and a single start and end events. The corresponding CSP script for this BPMN is shown in Figures 2-2 to 2-4.

```
datatype Msg = init | done
datatype Node = startFlow162 | Register_Applicant | Get_Docs |
agateFlow290 | agateFlow30 | Check_Requirements | Get_GAT |
Get_Attainment | Get_English_Test | Calculate_Ratio |
SMS_Notification_Of_Submission | Get_Submission | Nominate |
Send_Notification_Result | Get_Confirmation | Get_Cancellation |
Check_Status | Send_Ids | Send_Rejection | end0 | end1
channel starts : Node
channel fin , aborts , error , except : { 0, 1 }
channel Flow162 , Flow123 , Flow129 , Flow149 , Flow290 , Flow211 ,
Flow24 , Flow25 , Flow30 , Flow29 , Flow189 , Flow229 , Flow959 ,
Flow89 , Flow179 , Flow230 , Flow301 , Flow221 , Flow380 , Flow19 ,
Flow200 , Flow249 , Flow281 , Flow258 , Flow300
```

Figure 2-2: Data type and channels declaration of UQU admission example

Each BPMN element (tasks and gateways) is defined as a node in CSP. BPMN tasks represent a piece of work that achieved by the process, e.g., '`Register_Applicant'` is a task representing the procedure of collecting an applicant's information when it is executed. In order to be able to model task triggers, a channel `starts` of type `Node` is defined. For example, the event '`starts.Register_Applicant'` indicates that the task '`Register_Applicant'` has already started . The channels `fin, aborts, error, except` indicate that the process reaches the end state when it successfully terminates and this is represented by fin or does not terminate successfully due to abortion or error or exception which is represented by abort, error and except, respectively. The flow between BPMN elements is modelled as a CSP channel called `Flow` and it indicates that there is an incoming/ongoing flow(s) to the assigned element.

In CSP, any process must be declared in terms of alphabet. A process alphabet is a collection of all events a process can engage when executing. In the above CSP fragment, there is an alphabet declaration for every BPMN element; start and end events, tasks and gateways. `AUniversityAdmission(agateFlow290)` refers to the alphabet of the '`agate290'` in the university admission pool and indicates that the gateway AND-Spilt can engage the events `'fin.0', 'fin.1', 'Flow149', 'Flow290'` and `'Flow211'`, as shown in Figure 2-3.

```
AUniversityAdmission(startFlow162) = { fin.0, fin.1, Flow162 }
AUniversityAdmission(Register_Applicant) = { fin.0, fin.1,
Flow162, Flow123, starts.Register_Applicant }
AUniversityAdmission(Get_Docs) = { fin.0, fin.1, Flow123, Flow129,
starts.Get_Docs }
AUniversityAdmission(agateFlow290) = { fin.0, fin.1, Flow149,
Flow290, Flow211 }
AInsurance(end0) = {fin.0, Flow16}
```

Figure 2-3: Alphabet declaration for car insurance

`PUniversityAdmission` (BPMN-element) indicates the behaviour of that element in terms of its incoming and outgoing flows and the work it represents (for tasks in particular) within this pool. For example, `PUniversityAdmission(agateFlow290)`, in Figure 2-4, indicates that `Flow149` is its incoming flow followed by `'Flow290'` and `'Flow211'` as its outgoing flows are parts of the university admission pool. `PUniversityAdmission(Register_Applicant)` indicates that the `'Flow162'` is its incoming flow and then some work is done by this task represented by the event `'starts. Register_Applicant'` followed by `'Flow123'` as its outgoing flow.

```
PUniversityAdmission(startFlow162) = ( Flow162 -> ( SKIP ) ) ; ( [] i:{
0, 1 } @ ( fin.i -> ( SKIP ) ) )

PUniversityAdmission(Register_Applicant) = ( ( ( Flow162 -> ( SKIP ) )
; ( ( SKIP ) ; ( ( starts.Register_Applicant -> ( SKIP ) ) ; ( ( ( SKIP
) ; ( ( SKIP ) ; ( SKIP ) ) ) ; ( Flow123 -> ( SKIP ) ) ) ) ) ) ; (
PUniversityAdmission(Register_Applicant) ) ) [] ( [] i:{ 0, 1 } @ (
fin.i -> ( SKIP ) ) )
PUniversityAdmission(Get_Docs) = ( ( ( Flow123 -> ( SKIP ) ) ; ( ( SKIP
) ; ( ( starts.Get_Docs -> ( SKIP ) ) ; ( ( ( SKIP ) ; ( ( SKIP ) ; (
SKIP ) ) ) ; ( Flow129 -> ( SKIP ) ) ) ) ) ) ; (
PUniversityAdmission(Get_Docs) ) ) [] ( [] i:{ 0, 1 } @ ( fin.i -> (
SKIP ) ) )

PUniversityAdmission(agateFlow290) = ( ( ( Flow149 -> ( SKIP ) ) ; ( (
Flow290 -> ( SKIP ) ) ||| ( Flow211 -> ( SKIP ) ) ) ) ; (
PUniversityAdmission(agateFlow290) ) ) [] ( [] i:{ 0, 1 } @ ( fin.i ->
( SKIP ) ) )
```

Figure 2-4: University admission behaviour

Regarding the above mapping, we believe that it could be much shorter if it has been produced manually. Wong's tool has generated lots of `SKIP` processes within the script, specifically the part of modelling the behaviour of each BPMN elements as can be seen in Figure 2-4. It could be manually mapped following Wong's structure by providing: (1) the declaration of the data types and channels, (2) the declaration of the process alphabets and (3) the modelling of the elements behavior. Thus, the part that might be manually enhanced is number (3) above, which is the elements behaviour, by cutting down the number of `SKIP` processes.

Wong facilitates the verification capabilities of CSP, i.e. refinement, to reason about BPMN generic properties, such as deadlock freedom and divergence freedom as well as model checking the order properties of a workflow. They map the LTL property specification to CSP in order to model check a BPMN diagram against behavioural rules (Wong and Gibbons, 2009). However, we did not use their mapping in this work because

their purpose is to model check the BPMN against properties that capture the order specification among BPMN tasks. Whereas we are interested in the occurrence of the BPMN tasks to check their availability after adaptation.

2.2.3 Policies

StPowla (Gorton *et al*, 2007) is the approach that we choose as a foundation of our work – the main motivation is that of the existing work it is the only approach that allows for dynamic changes to instances of workflows without human intervention. The explicit separation of core workflow and variability in StPowla allows for easy changes to the variability requirements by end users.

StPowla assumes that tasks in the process are executed through services, but services by their very nature could be automatic software components or software artifacts that require human collaboration (note that this human interaction is different to the one we try to avoid: we would not like human input to be required for dealing with workflow variability, but workflow tasks themselves are often discharged by humans). The workflow can be expressed in any workflow notation and the variability is captured by policies.

StPowla introduces two types of policies: reconfiguration and refinement. The former policies are event-condition-action rules that are triggered by a specific event and when satisfying some condition, perform the specified action(s). Refinement policies deal with requirements for individual tasks and have been discussed extensively in (Montangero et al. 2011); reconfiguration policies allow for changes to the workflow structure and thus are relevant here. The reconfiguration policies proposed by (Gorton, 2011) provide temporary changes to the WF by adding, deleting, failing, aborting or blocking simple or composite tasks as well as operators. He defines the effect of E-C-A policies as "a *short-lived* modification to the workflow: the modification expires when the workflow instance ends and the same core process is used at the start of the next instance". The main policy actions are summarized in Table 2-2. The syntax of StPowla policies (Gorton, 2011) is defined as follows:

```
Polrule:: = appliesto location [when triggers] [if conditions]
do actions

triggers:: = trigger | triggers or triggers

conditions:: = condition | not conditions | conditions or
conditions | and conditions

actions:: = action | actions actionop actions

actionop:: = and | or | andthen | orelse
```

Recalling the university admission example, suppose a policy `'EnsureOtherRequirements'` for inserting a new task called `'Check_Heritage'` in parallel with `'Check_Requirements'`. The policy syntax is written as follow:

```
policy EnsureOtherRequirements is

        appliesTo UniversityAdmission

           when Check_Requirements.task_ended

            do insert(Check_Heritage, Check_Requirements, true)
```

Where this policy has no condition and true indicates that both tasks are running in parallel.

Table 2-2: Reconfiguration Functions

Reconfiguration Function	Description
Insert	Inserts a set of items into the workflow instance
Delete	Removes set of items from the workflow instance
Fail	Designates an executing workflow task instance as having failed
Abort	Designates an executing workflow task instance as having been aborted
Block	Delays a task's execution

Reconfiguration functions are called over E-C-A policies to take an action by modifying the workflow structure.

The reconfiguration functions we consider here are:

1. Deleting atomic/composite task in sequence/parallel with existing atomic/composite task or an operator.
2. Inserting atomic/composite task in sequence/parallel with existing atomic/composite task or an operator.
3. Deleting or inserting an operator branch

In addition to the above reconfiguration functions, we introduce 'Replace' as a complex reconfiguration function combining delete and insert functions. It enables the replacement of tasks either by deleting followed by inserting or vice versa.

The existing work of StPowla describes the policies available and addresses the structural correctness; it does not consider the desirability of actions in terms of the functionalities of business processes. As a result, the existing work would allow for a workflow to be reduced to an empty process or an arbitrary process to be constructed from nothing. In terms of a simple example: a taxi booking could be changed into a process ordering a washing machine. Our work will build on this, but aims to add the constraints that ensure sensible adaptations in terms of their desirability.

2.2.4 Ontologies

Ontologies have been used for knowledge engineering in various research communities. It is a modelling language that captures the knowledge in a certain domain. The knowledge is represented as concepts and their relationship. Ontologies facilitate syntactical and semantical verification. However, it is rarely used for the verification purposes in self-adaptive systems, semantical-based verification in particular.
An ontology consists of the concepts in a specific domain with their relationship. In practical terms, ontology development includes defining classes in a hierarchy, defining slots and values, defining instances and relationships.

OWL (Horridge *et al*, 2004) is a logic-based ontology language and it consists of classes, individuals and properties. OWL classes are sets that contain individuals and OWL individuals represent instances of classes in a certain domain. OWL Properties represent the relationship that links individuals. Figure 2-4 depicts an example of OWL ontology. It has two classes A and B. Each class consists of two individuals (e.g. class A has IA_1 and IA_2). These individuals are linked to each other through the properties P1 and P2.

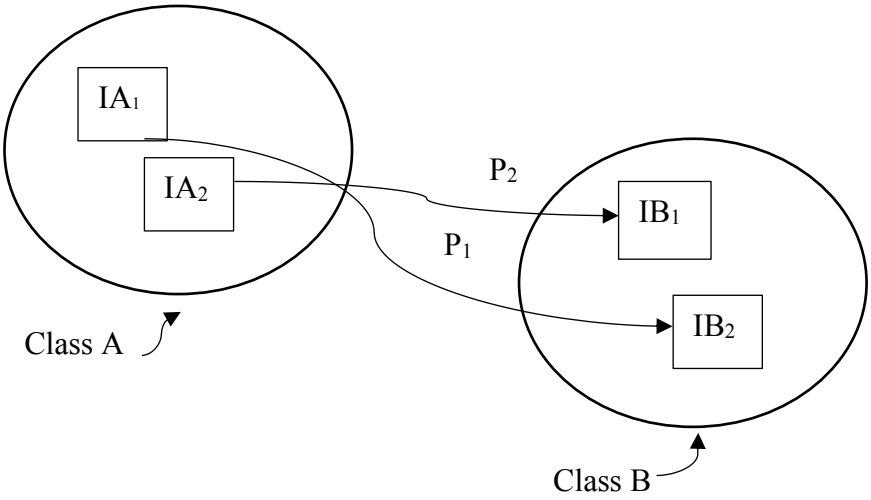


Figure 2-5: OWL ontology example

BPMN specification defines set of attributes associated with each activity. For example, task id, name and class to indicate the task id, name and its type between other BPMN elements. For every action in BPMN it is possible to define the related domain knowledge by capturing the BPMN components and defining a certain relationship between them to customize the ontology for a specific purpose.

There exist some approaches that aim to enhance BPMN semantics by developing BPMN ontologies (Abramowicz *et al*, 2012) and (Natschlger, 2011). (Natschlger, 2011) has developed a BPMN ontology using OWL (Web Ontology Language) and the ontology editor Protégé. She created two ontology files: 1) bpmn20base which contains a specification for all BPMN classes, their attributes, the relationship among individuals as

well as individuals and data values and the xml schema, and 2) bpmn20 which is an extension for the bpmn20base by adding the syntactical restrictions as defined by BPMN specification.

In this work, we use ontology in order to capture BPMN tasks as the domain concept due to the fact that BPMN tasks represent functional requirements and they are affected by policy actions and eligible for change. Ontology also facilitates defining extra vocabularies based on the domain semantic itself, in which it enhances our verification purposes when new functionalities are introduced at runtime.

2.3 State of the Art

This section discusses the related work moving from general preview on workflow flexibility to specific related challenges on providing assurances over flexibility and in more specific the challenge to ensure goal satisfaction in the face of abstraction and automation. We identify three research areas which concern the adaptation process as well as assurances in self-adaptive systems in general, and workflow systems in particular. They are: (1) Models@Run.Time and Business Process Models at Runtime (BPM@RT) (2) Requirements@Runtime and (3) Goal-Oriented Requirements Engineering for Business Processes (GORE-for-BP).

2.3.1 Workflow Flexibility

Although there is not an agreed definition for process flexibility, it can be seen as the ability of the workflow to react to uncertainty in a desired manner. (Regev and Wegmann, 2005) and (Nurcan, 2008) believe that flexibility combines both adaptivity and consistency, while the other consider correctness as a separate issue. Thus, among the variable definitions, represented in Table 2-3, we go for the definition provided by (Nurcan, 2008) in which flexibility dose not only mean adaptivity but also the quality of this adaptivity is part of flexibility. (van Der Aalst, Wil MP, Pesic and Schonenberg, 2009) consider flexibility as a synonym of adaptation, while (Reichert and Weber, 2012) identify adaptation as one of the various forms of flexibility. They define adaptation as the ability to temporarily deviate the flow during the execution of a BP. In the literature, various flexibility taxonomies exist and most of them share similar categorization, see Table 2-3. The advantage of those taxonomies is that they provide better understanding

of the notion of flexibility. Furthermore, they help to identify further investigations and research gaps in the area of workflow flexibility. As a consequence, we adopt the ideas from across the taxonomies that are relevant for our work without defining yet another taxonomy.

Adaptation can be achieved either at instance level or process level. The former considers temporary change per instance while the latter considers permanent change at process level. In this work, we focus on ad hoc change as it is achieved through E-C-A policies. (Reichert and Weber, 2012) provide a taxonomy for ad hoc process change. The taxonomy includes eight main categories and each category is refined into sub-categories. Below we classify the StPowla approach according to this taxonomy as it targets the ad hoc change.

1. Duration: the E-C-A policies provide temporary change
2. Degree of automation: the E-C-A policies reconfigure the structure automatically without user intervention
3. Scope: the E-C-A policies can change arbitrary region at any position
4. System control: not applicable as there is no access control for users, no concurrency or traceability control over policies
5. User interface: this is not applicable as the reconfiguration is automatically achieved
6. Subject: the E-C-A policies change the process structure, in particular the control flow schema
7. Specification: the specification of the E-C-A policies is change patterns
8. Correctness: the E-C-A policies provide syntactically correct adaptation.

Table 2-3: Definitions and taxonomies of workflow flexibility

Flexibility Definition	Taxonomy	Reference
Flexibility is the ability to yield to change without disappearing, i.e. without losing identity	No taxonomy provided	(Regev and Wegmann, 2005)

Flexibility Definition	Taxonomy	Reference
A business process is flexible if it is possible to change it without replacing it completely	• Abstraction level of change: Instance/Type • Subject of change: Functional/Organisational/ behavioural/ Informational/Operational • Properties of change: Extent/Duration/ Swiftness/Anticipation	[Regev et al., 2005]
Flexibility is the ability of the workflow process to execute on the basis of loosely, or partially specified model, where the full specification of the model is at runtime, and may be unique to each instance	• Dynamism • Adaptability • Flexibility	(Sadiq, Orlowska and Sadiq, 2005)
Business process flexibility is the capability to implement changes in the business process type and instances by changing only those parts that need to be changed and keeping other parts stable	• Abstraction level of change • Subject of change • Properties of change	(Regev, Soffer and Schmidt, 2006)
It is the ability to deal with such changes, by varying or adapting those parts of the business process that are affected by them, whilst retaining the essential format of those parts	• Flexibility by design • Flexibility by deviation • Flexibility by underspecification • Flexibility by change	(Schonenberg *et al*, 2008)

Flexibility Definition	Taxonomy	Reference
that are not impacted by the variations		
*Flexibility is considered as the capacity of making a compromise between, first, satisfying, rapidly and easily, the business requirements in terms of adaptability when organizational, functional and/or operational changes occur; and, second, keeping effectiveness	• Nature of flexibility: by adaptation, by selection • Nature of impact: Local/ Global • Nature of change: Ad hoc/Corrective/Evolutionary • Formalism • Transition • Versioning • Evolution techniques: Ad-hoc/Derivation/Reflexion/ Rule-based • Migration techniques: Cancellation/With propagation/without propagation • Flexibility Techniques: Late binding/Late modelling/The case handling	(Nurcan, 2008)

Flexibility Definition	Taxonomy	Reference
Flexibility includes variability (instance change at design time), looseness (instance change at runtime), adaptation (unforeseen change) and evolution (process change).	• Variability • Looseness • Adaptation • Evolution	(Reichert and Weber, 2012)
*Flexibility is the ability of an organisation to deal with both foreseen and unforeseen changes, and in consideration of the impact they can have on the BPs regulating the activities of the organisation	• Objectives: Variability/Adaptation/Looseness/ • Phases: Modelling/Analysis/Enactment/Monitoring • Languages: BPMN/BPEL/ECP/Petri nets/Process algebra • Mechanisms: Rules/Family of processes/Patterns/Modularity	(Cognini *et al*, 2016)

*Please note that those taxonomies are provided for studying the literature of workflow flexibility aiming at classifying the existing approaches and not to provide taxonomies for workflow flexibility.

A Workflow process's main characteristics that complicate enabling reliable flexibility in running processes are, as extracted from the literature:

- Dynamicity
- Complexity
- Uncertainty

- Knowledge-intensive systems
- Heterogeneous environments
- Dependency: dependency among workflow items, dependency between the five perspectives of the workflow (van der Aalst, Wil MP and Jablonski, 2000) , dependency between data and control flow and dependency between services and business processes
- Complex requirements (Chatzikonstantinou and Kontogiannis, 2016) .

Usually automatic adaptation techniques are related to the enactment phase in the process lifecycle. Examples of current approaches which automatically adapt workflows' behaviour are (Gorton, 2011), (Müller, Greiner and Rahm, 2004), (Sell *et al*, 2009) and (Burmeister *et al*, 2008) . Although automated adaptation is desirable, there are some approaches in the literature that dealt with flexibility in a manual manner (Reichert and Dadam, 1998), (Pang *et al*, 2011), (Hallerbach, Bauer and Reichert, 2010) and (Weber, Reichert and Rinderle-Ma, 2008).

In the context of this research, we consider the approaches that provide automated adaptation because we are interested in self-adaptivity and believe that the manual approaches do not meet the dynamicity and complexity nature of self-adaptive systems. In addition to the adaptation capability of these approaches, they provide quality over adaptation. However, they address quality in terms of syntactic correctness but neglect the semantic aspects as will be discussed in the following section.

2.3.2 Assurances in Adaptive and Self-Adaptive Systems

As discussed earlier, business processes should be flexible to deal with uncertainty that could be as a result of changing requirements, change in goals, laws and regulations, change in business context or environment. Due to complexity and uncertainty issues, workflow should be able to automatically and correctly react to change. Beside the needs for flexibility, workflow systems that are adaptable must be self-managing (Parashar and Hariri, 2005) . However, assurance is the key challenge in the automated adaptation systems (Tamura *et al*, 2013), (Cheng *et al*, 2014) and (Krupitzer *et al*, 2015) in order to guarantee desired behaviour and, therefore, prevent serious consequences. In the context of software engineering, it is argued that self-adaptive systems must be integrated with

self-testing capabilities (De Lemos *et al*, 2013b) in order to support on-the-fly verification.

There are two reasons why self-testing capabilities are desired (1) complexity and (2) online runtime adaptivity. Hence, on-the-fly verification, self-testing capabilities (De Lemos *et al*, 2013) and intelligent decision-making techniques are required. On-the-fly verification means that the workflows have the ability to meet runtime verification properties. Self-adaptive V&V include techniques and mechanisms for assuring correctness properties in self-adaptive systems.

Workflow correctness issues are identified in the literature as a multifaceted concept including three major classes: syntactic, behavioural and semantic. Each class encompasses a number of properties as shown in Table 2-4 (Tamura *et al*, 2013). The term correctness is used here to refer to assurance. According to a definition provided by IEEE Standard Glossary of Software Engineering Terminology, assurance is "a plan and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements" (Radatz, Geraci and Katki, 1990). Self-adaptive systems must be assured to satisfy both functional and non-functional requirements (Cheng *et al*, 2014).

Table 2-4: Correctness criteria for self-adaptive systems with related approaches

Correctness Class	Properties
Syntactic	• **Inheritance** (Van Der Aalst, Wil MP and Basten, 2002), (Rinderle, Reichert and Dadam, 2004) • **Compliance state** (Rinderle, Reichert and Dadam, 2004), (Reichert and Weber, 2012) • **Control flow correctness** (Reichert and Dadam, 1998), (Gorton, 2011), (Patig and Stolz, 2013)
Behavioural (Generic properties)	• **Exception handling** (Ali, 2012) • **Deadlock and Divergence freedom** (Aguilar *et al*, 2016) • **Liveness and safety** (Patig and Stolz, 2013)

Correctness Class	Properties
	• **Soundness** (van der Aalst, Wil MP *et al*, 2010), (Wong and Gibbons, 2009), (Hallerbach, Bauer and Reichert, 2009) • **Reachability** (Gorton, 2011) • **Loop tolerance** (Rinderle, Reichert and Dadam, 2004) • **Dangling state** (Rinderle, Reichert and Dadam, 2004) • **Parallel insertion** (Rinderle, Reichert and Dadam, 2004) • **Schema prefix** • **Safe state** (Rinderle, Reichert and Dadam, 2004) • **Changing the past** (Rinderle, Reichert and Dadam, 2004)
Semantic (Domain/context properties)	• **Tasks dependency** (Ly, Rinderle and Dadam, 2008) • **Tasks compatibility** (Ly, Rinderle and Dadam, 2008) • **Coexistence tasks** (Pham and Le Thanh, 2015) • **Data flow correctness** (Rinderle-Ma, 2009), (Trcka, Van der Aalst, Wil MP and Sidorova, 2009) • **Compliance rules** (Reichert and Weber, 2012), (Fdhila *et al*, 2015), (Kumar *et al*, 2010)

This results in what is so called 'Requirements-Aware and context-aware software systems'. Although context-awareness seems to be a semantic property of the software, it is actually discussed in the literature in terms of flexibility but rarely consider assurance issues (Saidani and Nurcan, 2007), (Wieland *et al*, 2007) and (YongLin and Jun, 2008).

As shown on the table, most of the current approaches have addressed the challenges related to syntactic and behavioural properties and rarely consider the domain-specific semantic properties. In this work, we focus on the last class defined in Table 2-4 as we address the assurance challenge that is semantic and domain compliant.

(Ly, Rinderle and Dadam, 2008) provide an approach to ensure semantic correctness in terms of tasks compatibility and tasks dependency. Similarly, (Pham and Le Thanh, 2015) defined additional semantic constraints to the previous approach which is called coexistence constraints addressing the relationship among the workflow tasks. Furthermore, ensuring compliance to business rules and correct data flow are critical issues in order to guarantee desirable business outcome. Some examples of related approaches are shown in Table 2-4.

However, all the above-mentioned approaches ignore the quality of the new requirements (inserted/deleted tasks) with respect to the goal that the workflow is designed for. This thesis fills in this gap by providing an approach to reassess the quality of the adaptation when new requirements are introduced at runtime.

2.3.2.1 Requirements@Runtime and Models@Run.Time

In this section, we discuss the idea of Requirements@Runtime and Models@Run.Time as they are well-known research communities for providing assurances over adaptation in self-adaptive systems.

Requirements@Runtime (Bencomo *et al*, 2010) is a research community aiming at dealing with requirements (functional and non-functional) as runtime entities. Approaches (Feather *et al*, 1998), (Bencomo *et al*, 2010) and (Pasquale, Baresi and Nuseibeh, 2011) address the need for adaptation at requirements level and reconciling system's behaviour in response to changing requirements.

Self-adaptive systems are distinguished by two characteristics: context-awareness and self-awareness. Context-awareness is the ability of the system to monitor its context and automatically evolve to address contextual changes. Context is defined as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" (Abowd *et al*, 1999). Requirements-aware systems are an example of context-aware systems where requirements are considered the main context category. E-C-A policies can be seen as requirements-aware as they apply change according to changing requirements at runtime. In the StPowla approach, the E-C-A policies could be enhanced with the context information in order to obtain reliable change that is context-aware.

Models@Run.Time (Blair, Bencomo and France, 2009) has similar research interests as Requirements@Runtime. It adopts Model-Driven-Engineering (MDE) techniques to deal with uncertainty at runtime with the focus on representing requirement models as well as other models that could support dealing with change at runtime. Requirements@Runtime defined as "causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective" (Blair, Bencomo and France, 2009).

Models at abstract level are the core components of M@RT and can be used for several objectives: (1) Adaptation, (2) Abstraction, (3) Consistency and conformance, (4) Error handling, (5) Monitoring, simulation, prediction and (6) Policy checking and conformance(Szvetits and Zdun, 2016). We focus on the consistency and conformance objective as our approach uses goal models to reconcile behaviour model with its goal during runtime adaptation.

BPM@RT (Redlich *et al*, 2014) is a method inspired by M@RT for shifting business process models at runtime in order to address runtime issues. They argue that the dynamicity of today's businesses requires business processes to be more dynamic and automated to handle and support adaptation through its life cycle. In order to meet these requirements, business process models should be moved from design time to runtime models, where models represent the system's state at any time and allow adaptation and reasoning mechanisms. A survey in (Szvetits and Zdun, 2016) discusses various approaches that using M@RT based on objectives, techniques, kinds and architectures when using models at runtime. (Cheng *et al*, 2014) argue that M@RT is a foundation for assurance of self-adaptive systems. It provides the required level of abstraction to reason about the adaptation behaviour at runtime effectively. Furthermore, M@RT is a modular way that facilitates the adaptation and verification requirements for dealing with the complexity and dependency nature of self-adaptive systems.

Among various M@RT approaches we focus on the consistency and conformance approaches, which use runtime requirements as well as goal models in particular. In the following section, we discuss GORE-for-BP the RE methodology for mapping RE activities to business processes' activities.

2.3.2.2 GORE-For-BP for Requirements-Aware Workflows

Goal satisfaction is a critical issue for ensuring correct adaptation, especially automated adaptation in self-adaptive systems. Goal specification, as explained above, captures functional goals representing the functional requirements of an organisation. A BPMN diagram is constructed from logically related tasks that are contributing to achieve the goal in question. These tasks are representing functional requirements which are derived from the goal specification. Hence, having explicit links between goal model and behaviour or requirement models, facilitates behaviours traceability and therefore measurement of the satisfaction of certain properties.

Requirements Engineering (RE) concerns the study of requirements elicitation, specification, analysing and monitoring. Goal-Oriented Requirements Engineering (GORE) is a RE approach focusing on the goal as the main artefact for requirements elicitation and monitoring (van Lamsweerde, 2004). A goal model represents high-level system requirements as operational objectives in a tree structure. Those objectives are then refined into low-level requirements represented in a behaviour model. GORE-for-BP is an emerging research area as a result of the growing awareness of supporting business processes design, analysis and development with goal models.

The core idea of GORE-for-BP is to map RE knowledge state into BP knowledge state (or vice versa) (Poels *et al*, 2013). Several benefits for the RE process result in better understanding of modelling and verification activities as goal models are used at design time for modelling purpose and at runtime for reasoning about system's requirements (Ali, Dalpiaz and Giorgini, 2013). The realization of the importance of integrating goal models with business process models has been studied in the literature varying in the focus, scope and mature as explained in (Poels *et al*, 2013). However, the current approaches related to BPMN assurance challenges before and after adaptation fit into one of the following scenarios:

1. Approaches that use goal models for modelling or reengineering purposes: these approaches address the problem of goal compliance at design time to ensure the consistency of the designed requirement model to its goal specification. Examples of such approaches include GV2BPMN (Santos *et al*, 2010)KAOS4SOA (Nagel

et al, 2013), GPMN (Jander *et al*, 2011), GO-BPMN (Greenwood, 2008), Go4Flex (Braubach *et al*, 2010) and (Guizzardi and Reis, 2015).

2. Approaches that use goal model for adaptation and verification purposes: these approaches address the challenges related to reconciling the behaviour of the requirement models in response to uncertainty in goal specification. In other words, they assume uncertainty at goal level and according to that the behaviour of the requirements models is adapted to satisfy the changing goals(Koliadis and Ghose, 2006), (Burmeister *et al*, 2008) AutoRelax (Fredericks, DeVries and Cheng, 2014).

The second scenario is more related to the work presented in this thesis as it concerns the goal-process satisfaction relationship in adaptive workflows. However, they reassess the requirements models in response to goal change while we reassess the requirements models against their goals in response to requirements change. Furthermore, these approaches do not provide a clear methodology on how the adaptation is achieved at behaviour level.

2.4 Requirements-Aware/ Goal-Compliance Business Processes

This research focuses on providing assurances over changing requirements to enable change only for goal-compliant requirements. The proposed approach is built on requirements-aware workflows that realise changing requirements through policies. It strengthens the awareness in the requirements-aware workflows to be aware of its original requirements represented in goal specification to guarantee the correctness of the new requirements.

In the previous section, we consider the current trend on providing assurances in self-adaptive systems including M@RT and GORE-for-BP. In this section, we clarify the link between our approach and the approaches in these research areas, however we do not build our work upon any of these approaches. Considering M@RT, this work uses runtime representation of the corresponding models (requirements and goal models) and proposes the link among them at abstract level. The abstraction of these models enables to define consistency properties for runtime verification. Table 2-5 shows the models representation at design time and at runtime.

GORE-for-BP facilitates the use of goal models for defining the satisfaction properties before adaptation and hence ensuring their satisfaction during requirement-based adaptation. In this work, we use the goal model with the BPMN to address the goal satisfaction when processes change their behaviour in response to changing requirements.

Table 2-5: Models@Run.Time and our approach

Design Time Models	Runtime Models
Goal Model: KAOS	• CSP Properties • Ontology-based properties
Behaviour model (source): BPMN	• CSP model for FDR verification • XML-based model for ontology verification
Adaptation logic: reconfiguration functions	• Java functions

2.5 Summary

This chapter presented the background concepts and techniques used by this research. It also discussed the trend in the field of self-adaptive systems considering two major challenges, adaptation and consistency, in the face of complexity and dynamicity. Furthermore, it identified the consistency challenges related to automated adaptation and proposed the solution provided by this research.

Chapter 3 Goal-Compliance Framework

In this chapter, we introduce our running example throughout this thesis as an illustration of our approach. We also give an overview of the proposed goal-compliance framework, the concept behind it and its implementation. We discuss its notable aspects and how it supports running instance change at business level. As the framework is enabling runtime functional change to workflows, we describe and discuss runtime infrastructure and properties and how this supports online workflow reconfiguration and verification. The proposed framework supports a number of important features of runtime adaptation including: change per instance, online adaptation (i.e., adapt running instances), automatic adaptation using E-C-A policies and runtime change management.

3.1 Proposed Framework: Overview

The goal-compliance framework supports 'on-the-fly' workflow adaptation while at the same time providing assurances on goal satisfaction during adaptation. The framework comprises different techniques and tools to facilitate and support runtime workflow adaptation and verification at business level.

We define three methods to validate workflow behaviour which is affected by policies to match the different ways that policies can adapt workflow behaviour by inserting, deleting and replacing tasks.

The first method 'goal-task dependency' is dealing with policies of type 'delete' by governing the policies effect on workflow behaviour according to goal-satisfaction properties. The second method is dealing with policies of type 'insert' whose effect on workflow behaviour is semantically different from policies of type 'delete'. The method is called 'domain-task conformance' and it controls the insertions of new tasks to the running workflow instances according to domain consistency properties. The third method is called 'task-task consistency' and it is defined as a result of analysing the effect of replacing existing tasks with new ones. Hence, it controls the effect of policies of type 'replace' on workflow behaviour. Although the aforementioned methods are different in the way they are working, they are designed to solve one problem, which is goal satisfaction in self-adaptive workflows. In other words, they guarantee that any functional change to workflow behaviour does not violate its original goal.

We assume the workflow specification is expressed in BPMN, saved as XML document for reconfiguration purposes. This is due to the fact that XML has a clear and consistent structure. The main focus of the proposed framework is the verification process as mentioned above. It validates the adapted instance against what we call goal-compliance constraints. We define three sub-categories of the goal-compliance constraints as shown in Figure 3-1. They are goal-task dependency constraint, domain-task compliance constraint and task-task consistency constraint. The methods to handle goal-compliance constraints are discussed in detail in Chapters 4 and 5.

The notable aspects of the proposed framework can be summarised as follows:

1. **Online workflow reconfiguration at instance level:**

The framework provides flexibility for workflow systems by inserting, deleting and replacing workflow tasks. The proposed framework provides automatic and online reconfiguration at instance level through E-C-A policies. Instance change (also known as ad hoc change) is an important requirement in today's workflow systems due to their dynamicity and uncertainty. Further details in this regard are discussed in Chapter 1.

2. **Goal-compliance verification capabilities in addition to syntactical correctness:**

The goal-compliance runtime verification is the key feature of the proposed framework. Before applying any workflow change, the framework has the ability to check the corresponding constraints and decide whether to accept the change or not. Each change has its corresponding constraints based on the analysis of its effect on goal satisfaction, as discussed above. Furthermore, syntactical correctness is achieved by reconnecting the flows among workflow elements.

3. **Using traditional verification techniques to produce new solution:**

Self-adaptive systems need dynamic and sufficient verification mechanisms to cope with the runtime environment. We use two different verification methods: trace refinement and ontology. The reason why we choose two different methods is because policies have semantically different impact on achieving the goal as they insert or delete new/existing

functionalities. The trace refinement enables an automatic verification among the adapted workflow and a set of properties. It is a stable method as the compliance properties are predefined according to the workflow in question. However, trace refinement is not applicable to verify change of type insert or replace as they might add extra functionalities (tasks) that are unknown prior to runtime. As a result, the framework benefits from the ontology for modeling and verifying domain semantics as the ontology can captures un/planned functionalities in a specific way for verification purposes. This point is explained in detail in Chapters 4 and 5.

4. **Facilitating extra semantic property checking "Reusability":**

We use ontologies to model workflow domain by defining the workflow domain concepts and their relationship. However, the ontology could also facilitate other types of semantic checking by enhancing/reusing the ontology to add more constraints or define different relationship among the domain concepts. Furthermore, it could be used for querying the ontology while performing such a semantic verification.

5. **Applicable performance in practice:**

The framework is built in Java and it uses different techniques and tools to perform reconfiguration and verification. However, the framework performance evaluation shows promising results as the execution time taken for reconfiguration as well as verification is reasonable for runtime. For example, it shows that the time taken to process a workflow of size 100 tasks is less than one minute. More discussion and implementation details regarding this point is explained in chapter 6.

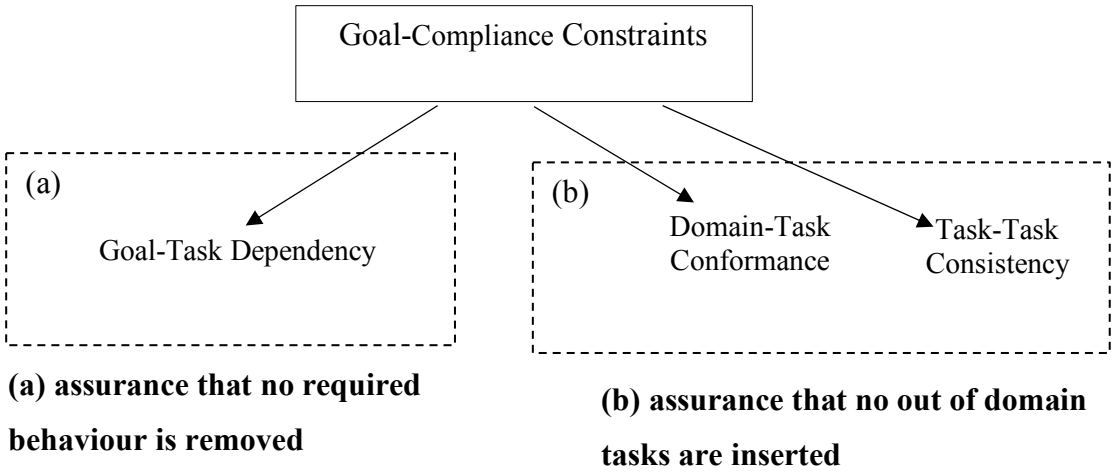


Figure 3-1: Verification requirements for different dependencies

3.2 Runtime Environment Infrastructure

The goal-compliance framework performs workflow reconfiguration and verification at runtime. Therefore, we give a brief description of the runtime environment infrastructure and how the proposed framework operates. Runtime verification is a well-known research area in the context of software engineering. While we implement runtime adaptation and verification, we give an overview of runtime infrastructure in line with the proposed framework.

The goal-compliance framework for self-adaptive workflows is supposed to be embedded at runtime environment which is mainly composed of workflow and policy engines. In the following, we give an overview of the main artefacts that manage, run and support self-adaptive workflows.

The workflow engine gets the workflow scripts as input which get triggered, run and then produce an output to a user. In the context of this work, workflows are adapted in response to E-C-A policies but the adaptation process is controlled by goal-compliance constraints. Figure 3-2 depicts the real-world runtime environment for self-adaptive workflows. It shows that workflow engine and policy server are communicating with each other. Workflow engine sends the triggering data and context and other data to policy server which actually does two things: (1) it has an interface with authorised users who deploy policies by adding or removing policies, which are stored in policy repository, (2) it evaluates policies based on the data received from workflow engine and decides whether they apply or not. Policy server has the enforcement point where the interaction happens

with workflow engine. In the literature, they differentiate between policy enforcement point and policy decision point whereas we do not see any difference because policies need to be where they are being enforced.

After the policy server gets the deemed data from the workflow engine, it looks up the repository for matching policies. First of all, it looks at which trigger data matches the triggers received from workflow engine. The trigger is an optional value and it describes the situation of a task, e.g., 'task Completed' and 'taskStarted'. Then, it looks up for matching conditions where they are true based on the context data retrieved from the workflow engine. Please note that any policy does not have a condition is considered to be true by default. Finally, all applicable policies are returned by the policy server which tells the workflow engine what to do based on the actions it gets from the applicable policies. So, the policy server now gets back to the workflow engine with one of these actions; insert or delete.

The workflow engine then changes the workflow instance to react to these actions if they are approved by goal-compliance constraints. We assume that goal-compliance constraints are coded as extra functionalities to the workflow engine. These constraints help to produce qualitative workflows while the absence of them might lead to syntactically correct workflows but not semantically. This is due to the fact that policies could delete, insert or replace anything at any position. For example, as mentioned above any policy with no conditions are evaluated to be true. In the context of our work, the workflow engine is responsible to apply the change to the running instance while ensuring its compliant to the goal-compliance constraints.

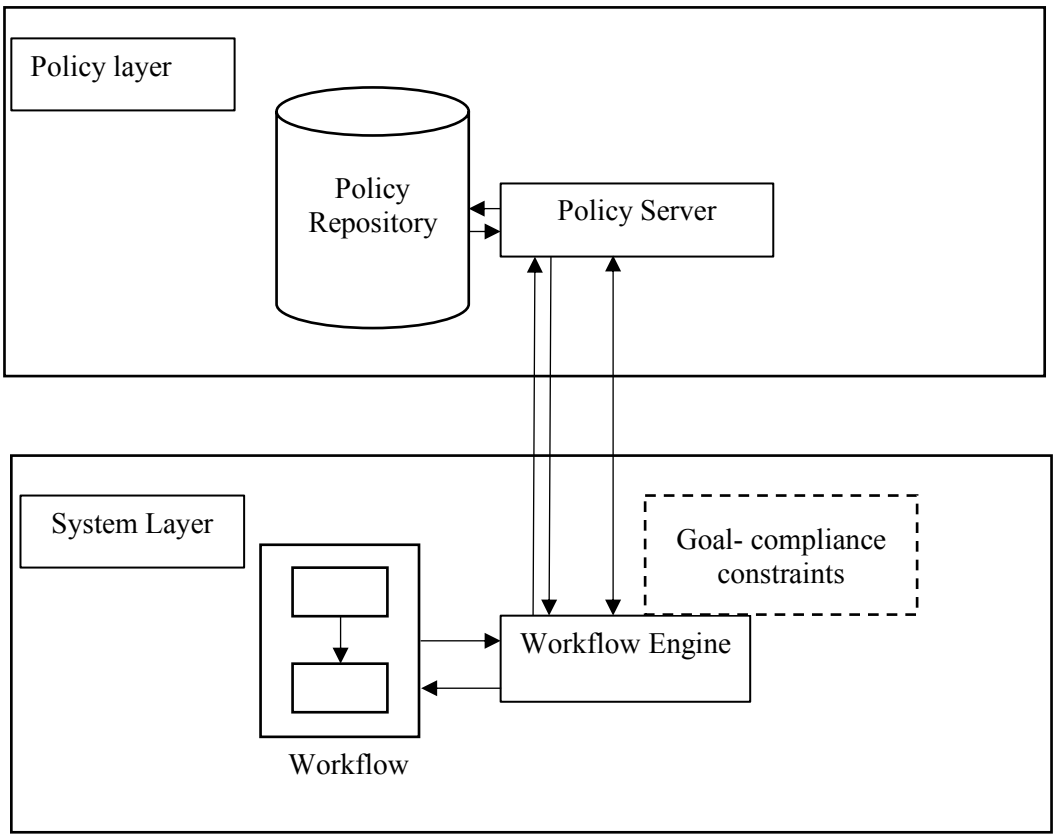


Figure 3-2: Runtime Environment for Self-Adaptive Workflows

3.3 Goal-Compliance Framework

The following subsections discuss the architecture and the implementation of the proposed framework.

3.3.1 Architecture

The goal-compliance framework as mentioned above performs its jobs at runtime with help and support from workflow engine and policy engine. The workflow engine facilitates the reading and running of workflow specification and managing instance reconfiguration. While the policy engine looks up the policy repository for matching policies as well as manages the deployment of applicable policies.

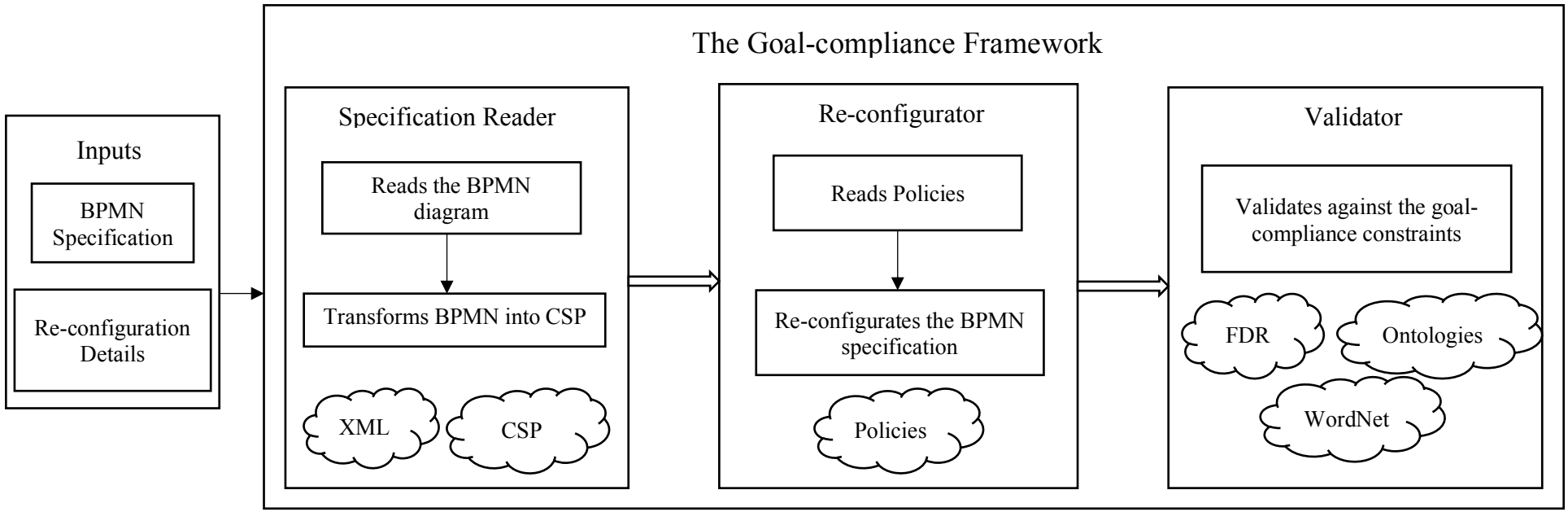


Figure 3-3: Goal-compliance framework: Architecture

As can be seen from Figure 3-3, the proposed framework reads the original workflow specification, the modification details as its inputs. The workflow specification is represented in BPMN (Business Process Modelling Notation) which is provided as an XML-based file. The modification details can be expressed in a configuration file, that specifies the changes to the workflow specification including the type of change (delete/insert), the position, the new/existing task Ids and the nature of the change (sequence/parallel). The proposed framework consists of three main components namely, Specification Reader, Re-configurator and Validator. The brief description of each one is provided below:

- **Specification Reader:** This component is responsible for reading the existing workflow specification and transforming it into an in memory state for fast processing and easy manipulation of the modification. This can be achieved by utilizing some XML interfacing APIs (Application Programming Interface).
- **Re-configurator:** The re-configurator is responsible for processing the actual change operations e.g. insertion of the new task into existing workflow specification. This component is responsible to interact with the XML and CSP files to perform the adaptation. It performs reconfigurations at XML level as it has a clear structure. It then invokes Wong's tool to transform it to CSP script if the reconfiguration is of type 'delete' for verification purposes. This transformation is carried out for verification purposes using FDR. The verification process for the other reconfiguration types; insert and replace; is performed through the interaction with ontology and WordNet (*WordNet, 2012*). So, the framework deals with the XML file in this case without the need to transform it to CSP.
- **Validator:** The validator is responsible for ensuring that the modification is according to the given specification and it doesn't violate any of the goal-compliance constraints, the constraints that are defined in the context of this research (see Figure 3-2) to provide assurances on goal compliance after requirements adaptation. This can be achieved by exploiting the FDR, ontology and WordNet.

The proposed framework works in the following sequential order:

1. Read the existing workflow specification
2. Read the reconfiguration details
3. Validate the reconfigurations
4. Reconfigure the workflow, if satisfying the goal-compliance constraints and produce adapted workflow specification
5. Otherwise, save the original specification

If the verification result was not successful, the framework neglects the reconfiguration request and continues to run the original specification or suspends the workflow execution and gives feedback for authorised users to take a decision.

3.3.2 Implementation

This section describes the details of the prototypical implementation of the proposed framework explained in the previous section. Java has been used as the main development language. Furthermore, specific Java libraries, such as OWL (*OWL API, 2015*) and Java WordNet Interface (*JWI 2.4.0., 2015*), are used for interfacing with ontology and WordNet dictionaries.

The framework reads all necessary information about current reconfiguration from an application configuration file which is called 'appConfiguration', see Figure 3-4. This file consists of two main configuration parts: Basic and Operations. The Basic part consists of the main parameters that manage input and output files, providing domain name, ontology files and WordNet dictionary paths. The Operations part holds all parameters of each reconfiguration operation. This part can hold multiple operations for the purpose of complex reconfiguration. Each operation is defined by providing its type, nature and other information. The operation could be atomic-insert, atomic-delete, composite-insert or composite-delete. The nature indicates whether the operation is going to apply the change in sequence or parallel manner within the running workflow instance.

The implementation of the proposed framework runs in a sequential manner and can best be described by the flowchart provided in Figure 3-5.

The brief description of each step of the above process flow is described below:

1. In the first step, the framework reads the workflow specification. The workflow specification is a BPMN document and is in XML format.
2. The workflow specification is the collection of various details like tasks, gateways and pools. The proposed framework converts these details to respective memory objects. The key reason of this conversion is to easily deal with them in the memory and to avoid file reading again. There is one Java class to represent each entity of workflow specification.
3. The configuration file consists of the details of workflow reconfiguration process (e.g. which kind of operation must be executed and what are the respective details). All the configuration is read at once and stored in memory as a java object.
4. Once the workflow specification has been read to memory and the required operation is determined, then the operation is executed and the workflow specification has been changed in memory.
5. The framework then validates the current change against the predefined constraints. This depends on the type of reconfiguration as discussed before. The framework uses different tools and techniques, when validating the reconfiguration, according to the reconfiguration type. If the reconfiguration was of type ‘delete’, it uses FDR for verification. If it was of type ‘insert’ or ‘replace’, the framework uses ontology or ontology and WordNet for verification.
6. In the last step, the modified workflow specification is saved to disk if the verification was successful. Otherwise, the original specification is saved. The framework then has the ability to either run the original workflow specification or stop.

```
<AppConfigurations>
     <Basic>
          <inputFile>BPMNFiles/PizzaExOut.xml</inputFile>
          <outputFile>BPMNFiles/Pizza_Out_1.xml</outputFile>
          <ontologyFile>BPMNFiles/PizzaDomain.owl</ontologyFile>
          <domainName>PizzaDelivery</domainName>
          <dictPath>/Users/budoorallehyani/Documents/WordNet-3.0/dict/</dictPath>
          <assertionFile>BPMNFiles/assertions.txt</assertionFile>
          <evalFile>BPMNFiles/eval1.csv</evalFile>
     </Basic>
     <Operations>
     <Operation sequenceID="1">
               <!---Operation types include display_only,atomic_seq_insert/atomic_seq_delete/composite_insert/composite_delete/atomic_par_insert/replace -->
               <operationType>atomic_seq_insert</operationType>
               <!--gateway Name and Type needed for parallel Insertion-->
               <gatewayName>OR</gatewayName>
               <gatewayType>OR</gatewayType>
               <!-- nature of operation such as SEQUENCE/PARALLEL -->
               <nature>sequence</nature>
               <!--new Task Id-->
               <taskId>Add_Side</taskId>
               <!--Composite task sub components include (Start Items, End Items, Tasks, Gateways, Flows-->
               <startIds>Car_Risk</startIds>
               <endIds>cEnd</endIds>
               <subProcTaskIds>cA,cB,cD</subProcTaskIds>
               <!-- Flow format (flowID-fromId-toId) -->
               <flowIds>spFlow1-cStart-cA,spFlow2-cA-cB,spFlow3-cB-cEnd</flowIds>
               <!-- Composite task sub components finished. -->
               <!-- Existing task Id -->
               <existingTaskId>Order_Pizza</existingTaskId>
          </Operation>
     </Operations>
</AppConfigurations>
```

Figure 3-4: Example of the appConfiguration setting file

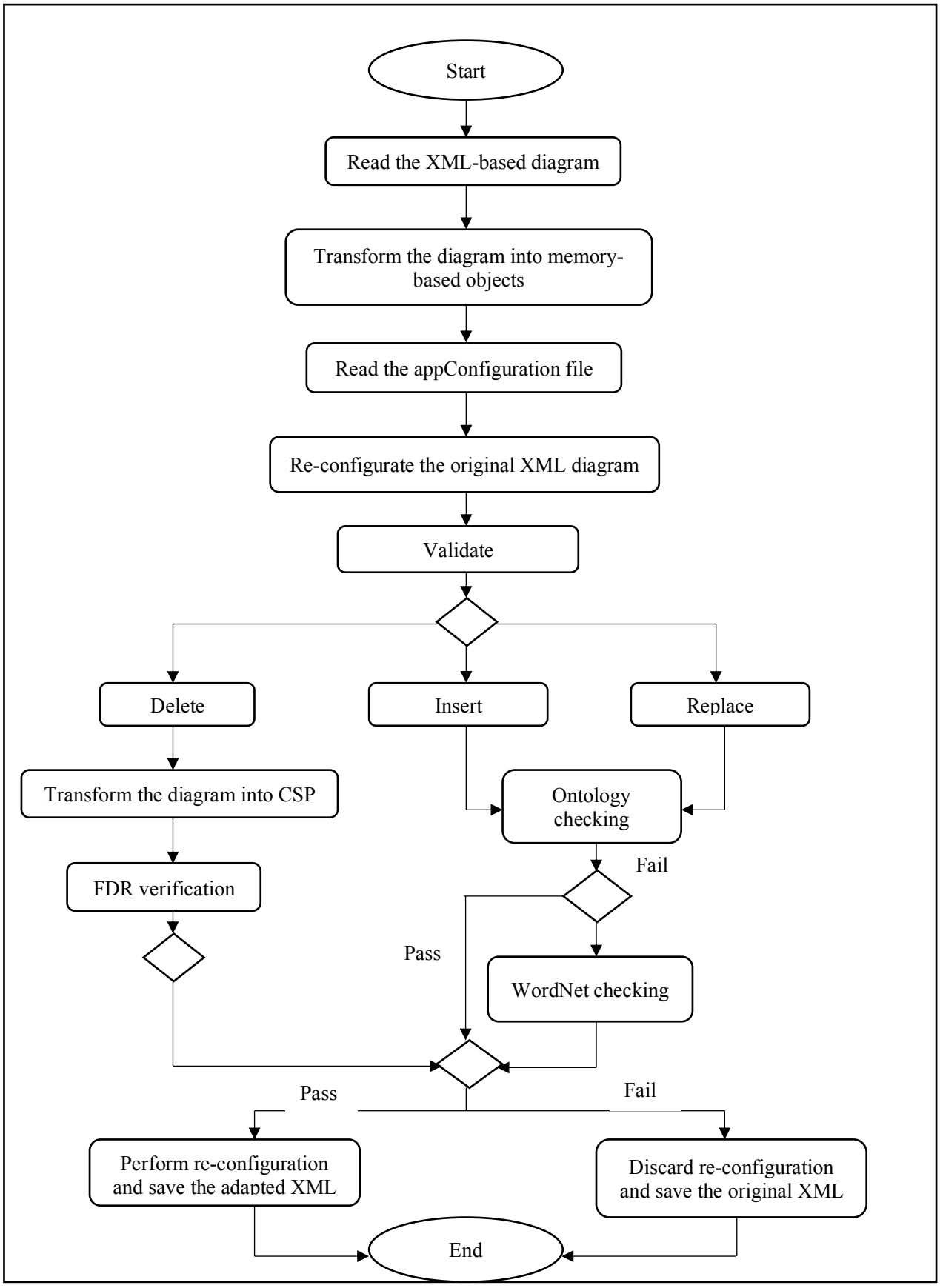


Figure 3-5: Algorithm 'flowchart' for the goal-compliance framework

The above steps are executed through the help of various functions and Java classes. The following are the most important classes and their brief description:

1. **BPMNController:** This class is the main class and it contains the main function. It reads the configuration file and create an object of that file to pass it to Diagram class, where all other functions are performed.
2. **Diagram:** This class is considered the orchestrator of the framework and all the steps mentioned in Figure 3-6 process flow is triggered from the various functions of this class.
3. **Operation:** This is a data class, which is the corresponding representation for the relevant details of the reconfiguration operation i.e. Insert, Delete and Replace.
4. **DiagramEntity:** This class represents the structure of any BPMN entity: BPMN pool, start and end events, activities, flows and gateways. All other entity classes are derived from this class.
5. **UtilOntology:** This class basically provides interfacing with OWL API to interact with the Ontology file. This class contains all the necessary operations that is needed for insert and replace reconfiguration operations.
6. **UtilWN:** Similarly, to UtilOntology class, this class provides interface with JWI library for the interaction with WordNet dictionary at runtime.
7. **AppConfiguration:** This is a data class, which contains functions to read the configuration file and store all the necessary parameter settings.
8. **BasicConfiguration:** Reads all necessary information from the appConfiguration file at runtime and specifically from the Basic part.
9. **OperationConfiguration:** Reads all necessary information from the appConfiguration file at runtime and specifically from the Operations part.
10. **XMLHelper:** This class is the handler to interact with XML (i.e. BPMN) file and contains methods like read workflow specification and write modified workflow specification.
11. **PerformanceEvaluation:** This class holds and manages all the evaluation functions which are explained later in chapter 6.

3.4 Summary

This chapter presents the proposed framework, its features and implementation. We discussed the runtime environment where the framework is supposed to perform its functionalities. Nevertheless, the behaviour of policies and how they run and managed through policy and workflow engines are explained in addition to the characteristics of runtime verification in the context of our research. Next chapters, 4 and 5, are going to comprehensively discuss the three methods supported by our proposed framework.

Chapter 4 A Refinement-Based Approach for Delete Policy

In this chapter, we show how deleting a workflow activity would have a serious impact on achieving the business outcome. To that end, we define a goal-task dependency constraint as well as satisfaction function to preserve workflow goal during runtime adaptation. Specifically, we analyse the impact of removing BPMN activities on goal satisfaction by establishing a management link between functional objectives in goal models and functional requirements in BPMN models. This link is formally identified and a verification mechanism exploiting the CSP formalisms and tools is introduced.

4.1 Adaptation with Delete Policy

The framework allows the deletion of sequential atomic, parallel atomic and composite tasks. The change affects only the running instance and expires once it terminates. In addition to the adaptation capabilities, E-C-A policies have no semantic constraints that govern their behaviour at runtime. Policies can delete anything from the process neglecting the goal the workflow is designed for. Thus, policies might require the removal of activities that contribute to the achievement of the business goal. This may cause undesired behaviour violating the organisation's goals and assumptions.

The policy syntax for deleting a task is (as defined in (Gorton, 2011)): `Delete (T)`, where `T` represents the task to be deleted.

The deleted tasks can be atomic sequential, atomic parallel and composite. The policy engine guarantees the correct behaviour in syntactic terms by reconnecting the flows where the policy takes place.

Consider the UQU admission diagram in Figure 3-1: suppose policy R1 is deployed to delete the task '`Get_English_Test`'. This task is proceeded and followed by the complex gateways. When a policy is applicable to delete `'Get_English_Test'`, the policy engine takes care of the flow correctness.

4.1.1 Deleting a Subprocess

BPMN subprocess is simply a compound task that encompasses group of tasks contributing to achieve business outcome. Hence, deleting a subprocess means removing a block of tasks from the process model. In this context, we deal with subprocesses the same way as simple tasks. The goal-related subprocesses are identified through the establishment of goal-task dependency link and captures the availability of each task individually.

4.1.2 Deleting an Operator

BPMN gateways are used to control BPMN flows either by branching, merging, forking or joining (*OMG, 2013*). Unlike BPMN tasks, they do not hold any actual work. Exclusive OR (XOR) is used to create alternative paths within the process and only one path can be executed. Data-Based XOR and Event-Based XOR are used for the same purpose as XOR but the former executes only one path based on available data while the other one executes only one path based on event. Inclusive OR is also used to create alternative paths but one or multiple paths can be taken. Complex gateway is used to model complex synchronisation behaviour and one path can be taken based on the condition result. Parallel gateway (AND) is used to execute more than one branch in parallel.

From the above overview about BPMN gateways we study how deleting a gateway affects goal satisfaction. Goal specification indicates what and how to achieve business goals. What to achieve is represented by BPMN tasks, as explained earlier. How to achieve is related to the relationship among tasks either sequenced order or branched with any type of the above gateways. Task-Task relationship is indicated by the order properties, i.e., task A is executed before task B and this is represented in the BPMN diagram by their flows. The second type of relationship which is gateway-related tasks indicates parallel execution of tasks or one/multiple execution of tasks depending on the gateway. Goal model contains goals specification and their relationship. We neglect how their corresponding tasks are related in the process model as we focus on the occurrence of tasks but not the order between them.

Before deleting an operator, it must be clear that this operator does not violate the desired behaviour defined by business rules.

In the following sections, we analyse and discuss the impact of the delete policy on satisfying the original process requirements which are represented by goal specification.

4.2 Impact on Goal

Goal specification indicates an organisation's operational goals which are refined to at least one activity in the behaviour model. As a result, there is a strong link between goal and process models that cannot be ignored during adaptation. The process model captures the functional requirements of an organisation which have originally stemmed from its goal specification. Considering this 'dependency' link between goal and process models, we study the effect of removing a workflow task (functionality) in respect with the goal in question.

As the process specification captures the functional goals, those functionalities must be available at every process execution in order to satisfy their original goals. Otherwise the goal in question would not be achieved as expected and this might affect the process outcome which in turn might have a severe impact on both the organisation and its users. Table 4-1 shows an example of the goal-compliance rule and how policies can break it.

Table 4-1: Example of policy impact on goal satisfaction

Policy	Constraint	Organisation-specific goal example	Policy example	Policy impact example
Delete	Some workflow functionalities must always be available	The customer must receive a tracking number G: TrackingNumberSent	Delete the task 'Send_Tracking_Information'	**Desired behaviour (reject to delete the task):** The customer always receives a tracking number **Undesired behaviour (the task is deleted):** The customer never receives a tracking number

Recalling back our UQU admission example, we identify some variations to the core process which can occur in response to emerging laws or regulations, customer satisfaction or changing requirements. We assume these variations are applicable, under certain conditions, to some instances of the university admission process. The expected changes to the process are temporary and are applied to some process instances. We also assume that complex changes could be applied to a single instance at a time. For example, deleting a task and inserting a new one to the running instance. As in real world multiple instances could be run at the same time, we also assume that multiple changes might be applied to multiple instances at a time.

Table 4-2 shows an example of a policy that require deleting a task from the UQU admission workflow along with its syntactical definition and the reason of change.

Table 4-2: Expected policies in the UQU admission system

Policy action description	Syntax	Reason of change
R1: Delete the task 'Get_English_Test'	Delete (Get_English_Test)	Applicants' group

The university admission process has the task 'Send_Nomination_Result' which is contributing to achieve the goal 'NominationSent'. Thus, this task must be available at every workflow execution in order to guarantee the satisfaction of its corresponding goal unless the goal is changed. Otherwise, the workflow might deliver an undesirable outcome.

As discussed earlier, in Chapter 2, GORE-for-BP leverages the specification as well as the verification of adaptive business processes using the goal concept. In order to be able to study the effect of the adaptation process on satisfying the goal, we adopt a GORE methodology that supports the analysis as well as the verification of the adaptation process (Poels *et al*, 2013). GORE methodology was selected for creating a requirements-aware BPMNs for two reasons. Firstly, we believe that BPMNs are designed to achieve business outcomes; therefore, they capture functional requirements and any change to these requirements should not violate the original goal. The balance between requirements changing and goal satisfaction is a challenging issue due to a number of challenges related to the nature of self-adaptive systems including uncertainty, complexity and situation-specific cases.

Secondly, having explicit and formal goal specification support business analysts to reason about it at runtime. In particular, we adopt KAOS methodology from RE and link it to the BPMN requirements model. The aim of linking BPMNs with their goal models is to be able to decide which tasks (functionalities) are assigned to fulfil the goal (dependency relationship). This leads to the establishment of a management link between functional requirements and goal specification (or vice versa). Hence, the ability to trace functional requirements during adaptation guarantees that policies are not allowed to remove any workflow item aligned with a goal.

We consider goal specification in KAOS (Van Lamsweerde, 1991) that only captures functional requirements, the non-functional requirements are outside the scope of this study. The functional requirements are encapsulated through tasks in BPMN process models. The achievement of functional goals must be satisfied before and during adaptation. The satisfaction before adaptation means that the software is correctly designed according to stakeholders' goals (at design time). Whereas, during adaptation indicates that when the software changes its behaviour in the midst of its execution, it still satisfies its original goal (at runtime).

4.3 Goal-Task Dependency (GTD) Constraint: Linkage Specification

Although the dependency relationship between BPMN requirements models and their corresponding goal models is defined in the literature, to the best of our knowledge no study has facilitated this link to ensure goal satisfaction when process models are adapted. Instead, it is used to design a goal-compliance process model at design time or to adapt goal specification at runtime and accordingly adapt the process model. In the following, we illustrate how this link is defined.

4.3.1 From Logical KAOS Specification to CSP Properties

We propose the goal-task dependency constraint whose purpose is to find a dependency link to trace the satisfaction of the high-level goals when workflow components, i.e. tasks, are adapted. Workflows represent requirement models as they capture the functional requirements in terms of tasks, while goals represent the desirable functional requirements in terms of objectives. Therefore, each objective is captured through a task or group of tasks. The goal-task dependency constraint expresses that a task is dependent on a goal; i.e., this task is contributing to the achievement of this goal or objective. Nevertheless, business goals must be achieved during their entire life cycle. Defining the goal separately from the requirement model gives us the ability to verify the changed requirement model against its goal.

We defined a methodology for specifying and verifying the goal-task dependency constraint as follows:

1. Assign all goal-related tasks based on contribution relationship, which task (i.e. functional requirement) is contributing to achieve which goal (functional goal). This is based on requirements elicitation process and could be decided from the task id and the goal id as they suggest each other (Koliadis and Ghose, 2006).
2. Define the goal property specifications using the results from (1). Property specifications should state the availability (occurrence) of all goal-related tasks and must be consistent with the goal specification; i.e., the refinement between objectives should be reflected on property specification among their corresponding tasks.
3. Convert property specifications from (2) into CSP property specifications which states the availability of all goal-related tasks. We neglect temporal properties and focus on occurrence properties as we aim to compare the trace of the adapted process with goal properties in order to check goal satisfaction.
4. Check the trace refinement relation (satisfaction function) between goal properties and adapted process with FDR: `Spec [T= P`; which indicates that the process specification (`P`) satisfies the goal properties (`Spec`) that are extracted from goal specification.

The identification of this dependency constraint (from step 2) is based on the refinement relationship among objectives at goal level. Goal-related tasks are tasks that are related to objectives and they must be executed at all times unless the goal is changes. As a result, if all tasks are refined to objectives then deleting any task from the process is undesired. Goal-related tasks are thus inferred from the goal specification.

If there exist tasks that are not related to any objective, this means they are not contributing to the achievement of any objective, then removing them from the process will not affect its outcome. This could be the case when the BPMN specification is detailed or an instance does not require the execution of some tasks.

Each goal can be translated down to a single task or a group of tasks. Similarly, each task can be related to a single goal or a group of goals. However, if a task is related to more than one goal, this means that this goal is a sub-goal from a bigger goal. Nevertheless, it depends on the stakeholders' requirements and the way they specify the goal.

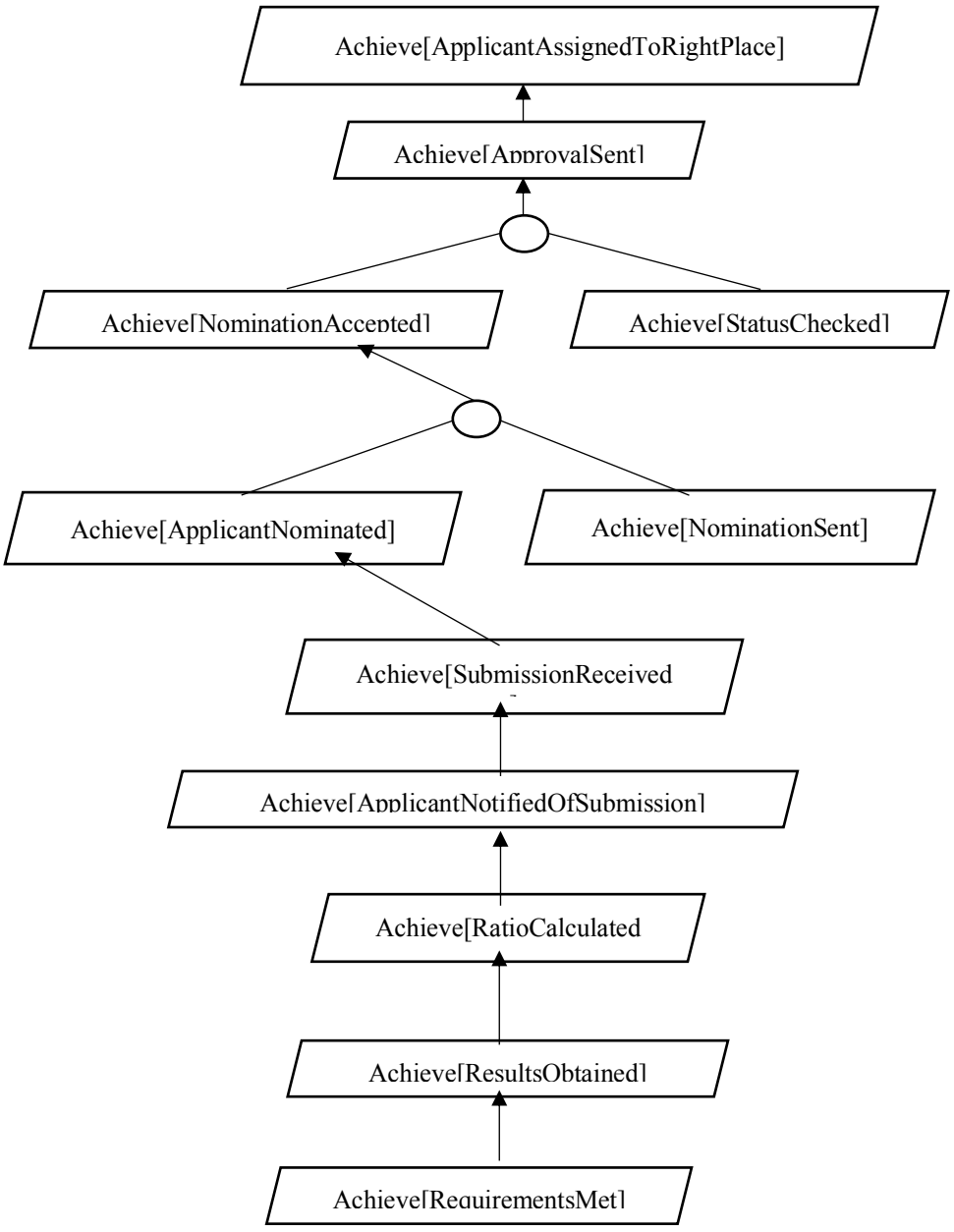


Figure 4-1: The KAOS goal model for the university admission process

We find two types of dependency when we define the goal-task dependency link between the goal model and the process model as follows:

1. A single task is contributing to the achievement of a single objective.
2. A Group of tasks is contributing to the achievement of a single objective and this could be:
   a) OR-grouped tasks
   b) AND-grouped tasks

c) Sequenced tasks

d) Randomly ordered tasks.

Based on the above discussion, we define CSP property patterns that guarantee the availability of goal-related tasks. The first dependency indicates a one-to-one relationship and is mapped to the LTL global universality pattern: □T. This means the task T may occur throughout the process execution. The second dependency indicates a one-to-many relationship and is defined as CSP patterns as follows:

a) The global occurrence of T ∏ T

b) The global occurrence of Ts; this case is treated as the first case because CSP is a primitive language and there is no notion for the AND operator

c) The global occurrence of T; ….; Tn (where ';' indicates T1 then T2, the sequence operator in CSP)

d) The global occurrence of Ts, as Ts are not in a sequenced order.

Table 4-3: Goal-task dependency for the UQU admission system

Goals	Tasks
'ApprovalSent'	'Send_Ids'
'NominationSent'	'Send_Nomination_Result'
'NominationAccepted'	'Get_Confirmation'
'StatusChecked'	'Check_Status'
'SubmissionReceived'	'Get_Submission'
'ApplicantNotifiedOfSubmission'	'SMS_Notification_Of_Submission'
'ApplicantNominated'	'Nominate'
'ResultsObtained'	'Get_GAT' and 'Get_Attainment' and/or 'Get_English_Test'
'RatioCalculated'	'Calculate_Ratio'
'RequirementsMet'	'Check_Requirements'

The KAOS goal model for the UQU admission example, depicted in Figure 4-1, includes only the functional requirements of the travel planning business process with the

refinement relationship among them. It does not include non-functional goals or expectations because they are not captured in the process model.

Goal properties are defined according to the relationship among goals in the goal model and the goal-related tasks. For example, the goal 'NominationSent' is achieved through the contribution of the task `'Send_Nomination_Result'` within the process and its satisfaction affect the satisfaction of the goal 'ApprovalSent' as the satisfaction of 'NominationSent' and 'StatusChecked' is important to achieve 'ApprovalSent'.

- `ApprovalSent= starts.Send_Ids -> SKIP`
- `ConfirmationReceived= starts.Get_Confirmation -> SKIP`
- `StatusChecked= starts.Check_Status -> SKIP`
- `NominationSent= starts.Send_Nomination_Result -> SKIP`
- `NominationAccepted= starts.Get_Submission -> SKIP`
- `ApplicantNotifiedOfSubmission= starts.SMS_Notification_Of_Submission -> SKIP`
- `PResultsObtained= starts.Get_GAT -> SKIP`
- `RResultObtained =starts.Get_Attainment -> SKIP`
- `QResultObtained=starts.Get_English_Test -> SKIP |~| SKIP`
- `RequirementsMet= starts.Check_Requirements -> SKIP`

The satisfaction of all goal properties leads to the satisfaction of the strategic goal the process designed for, which is 'ApplicantAssignedToRightPlace'.

4.4 GTD Algorithm: Verification

We choose the CSP as a specification language for different reasons: 1) available mapping from BPMN into CSP, 2) tools supported, 3) specification and verification language at the same time (i.e., the process and goal-task dependency properties are defined in CSP, 4) The notion of trace refinement is one of the main reasons for choosing CSP as it helps with the analysis and runtime reasoning.

The process model we have is expressed as a BPMN diagram and this BPMN is transformed to CSP using Wong's tool, as explained in Chapter 2. Goal specification is expressed in LTL patterns, so those patterns can be converted to CSP specification

manually following the goal-task dependency link specification. The constraint formulae then can be automatically checked using the FDR tool.

The delete process of the proposed framework refers to the ability to modify a given workflow specification through allowing the deletion of existing task(s).

The detailed flow chart of the GTD algorithm can be seen in Figure 4-2, while a brief description of the main steps is provided below:

1. The framework reads the workflow specification file and the configuration file that contains information on the task that is to be deleted. The framework first ensures that the task to be deleted exists in the specification.
2. The framework then ensures that the deletion operation does not violate any of the domain compliance constraints. If not, then the requested task is deleted from the in-memory representation of the workflow specification. A modified workflow specification must be produced at the end of the process.
3. If the deletion of the task violates any of the domain compliance constraints, then the framework will not allow the task to be deleted.

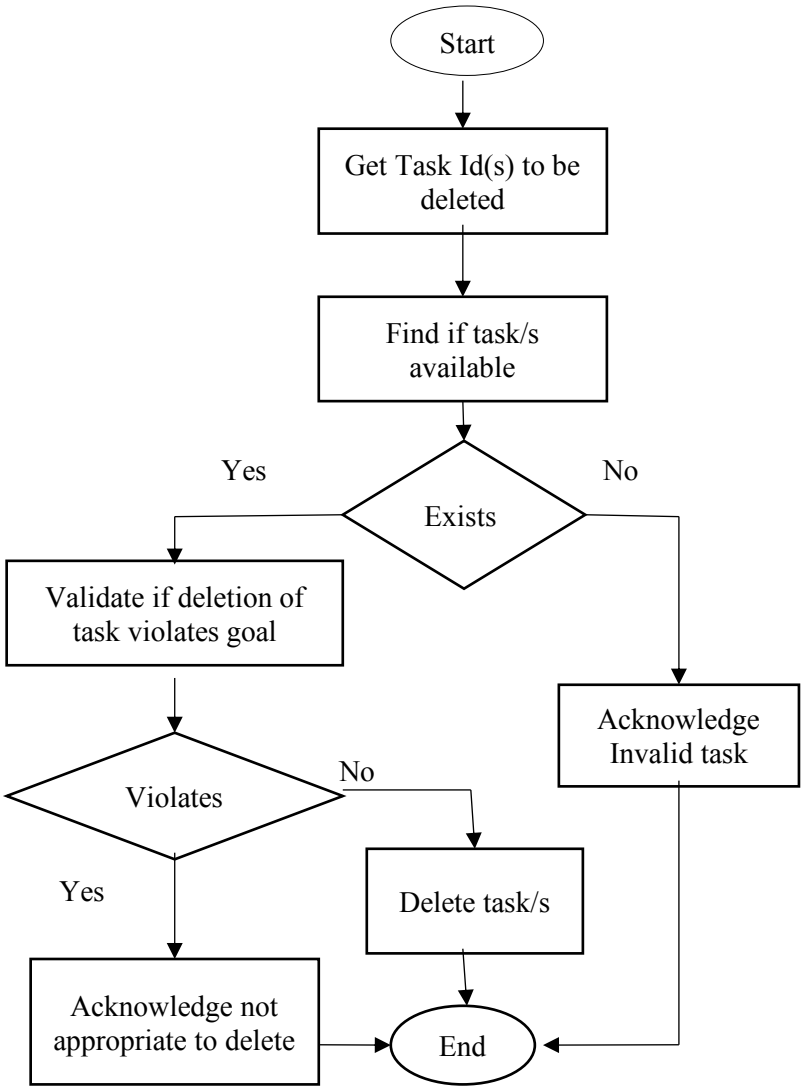


Figure 4-2: Flowchart of the goal-task dependency verification

4.4.1 Goal Satisfaction

We define goal-compliance properties based on the original goal specification in order to check their satisfaction. The satisfaction of all constraints is crucial for an instance to fulfil the strategic goal. However, goal properties specification is an upfront step as it is defined at design time. Therefore, the goal properties are defined and appended at runtime with the BPMN-CSP file and are processed all together with every instance checking.

Thus, the satisfaction of the goal properties leads to goal satisfaction. Each goal property is transformed into a CSP property specification. At runtime, each CSP property specification is checked against the adapted process through trace refinement assertion in FDR. If all assertions are successful, the model is consistent with its goal. Otherwise, the

model deviates from its goal specification and therefore the change is rejected. In this case, the process continues to run its original specification or stop to allow for user intervention.

Table 4-4: CSP assertion definition according to property types

Property type	CSP property specification	CSP assertion
1. One-to-one Task Ti to satisfy Objective O1	`P= Ti -> SKIP`	P [T= B \ (all other tasks)

2. OR-related tasks
 	
Ti OR Tj to satisfy Objective O2
	`P= let`
	`Sprc0= Ti -> SKIP`
	`Spec1= Tj -> SKIP`
	`Within Spec0 ∏ Spec1`
		`P [T= B \` (Ti or Tj and all other tasks)

3. AND-related tasks Ti AND Tj to satisfy Objective O3	`P= Ti -> SKIP` `PP= Tj -> SKIP`	`P [T= B \` (all other tasks) `PP [T= B \` (all other tasks)
4. Sequenced tasks Ti followed by Tj to satisfy Objectve O4	`P= Ti; Tj -> SKIP`	`P [T= B \` (all other tasks)
5. Randomly ordered tasks Ti, ….., Tj	`P= Ti -> SKIP` `PP= Tj -> SKIP`	`P [T= B \` (all other tasks) `PP [T= B \` (all other tasks)

As the focus is on occurrence of certain tasks, we use FDR trace refinement to compare traces between goal-properties and the adapted process. The property trace captures goal specification, where goal labels are matched to lower level task labels. For example, the goal `'ApplicantNominated'` is mapped to the task `'Nominate'` and the property captures the universal availability of this task. The adapted process trace represents the

process after adaptation and it must satisfy the property specification. If the property states the availability of a certain task is mandatory, the trace of the adapted process must perform this task.

Table 4-4 shows the mapping of properties to CSP specification and their corresponding assertions. FDR calculates these assertions under trace refinement to check their validity at runtime. One-to-one property is specified simply in CSP as a process that has just the task label in its trace, which is contributing to the achievement of an objective. Then, the assertion compares the trace of both the property and the adapted model by hiding all tasks from the trace of the adapted model except the task in the left-hand side. The CSP 'hide' operator enables certain tasks to be hidden from the process trace and this facilitates investigation of the model trace without some tasks.

The second property indicates OR-related tasks that are contributing to the achievement of an objective, so tasks are modelled in CSP specification using nondeterministic choice.

The goal-compliance framework now verifies the above defined goal properties through FDR trace refinement. In order to do that we provided the FDR assertion list that assert the trace refinement of the properties against the adapted process. From the above properties, we choose the `'QResultObtained'` to show how its corresponding assertion is written as it is concerned with the delete policy. The property `'QResultObtained'` is defined locally in the BPMN-CSP specification file as follows:

- `QResultObtained=starts.Get_English_Test -> SKIP |~| SKIP`

As R1 deleting `'Get_English_Test'`, this adaptation is expressed in CSP assertions by hiding this task from the original model specification along with all other tasks, as explained above. The CSP hide operator means show the trace without certain tasks.

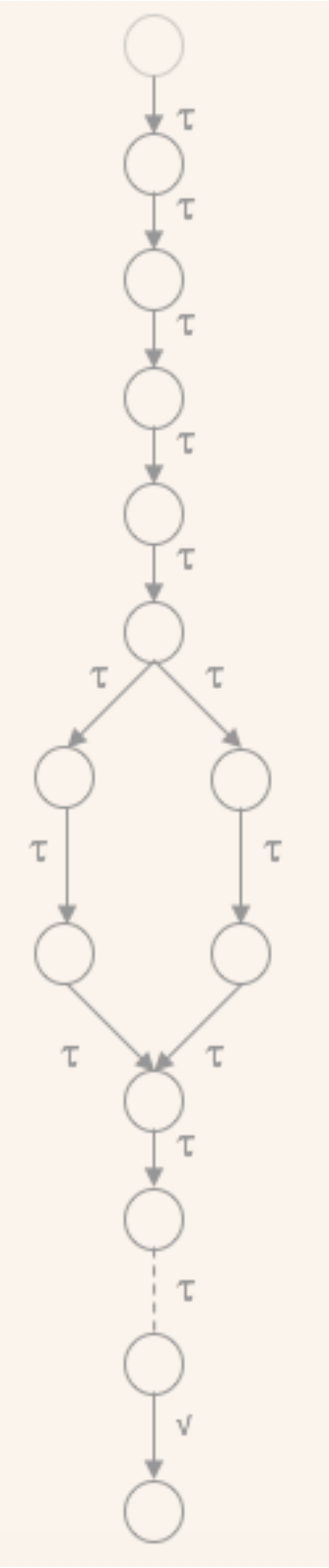
The assertion of the above property is shown below:

- ```
  assert QResultObtained [T= UUniversityAdmission
  \ {(starts.Register_Applicant, starts.Get_Docs,
  starts.Check_Requirements, starts.Get_GAT,
  starts.Get_Attainment, starts.Get_English_Test,
  starts.Notify_Of_Submission, starts.Get_Submission,
  ```
  ```

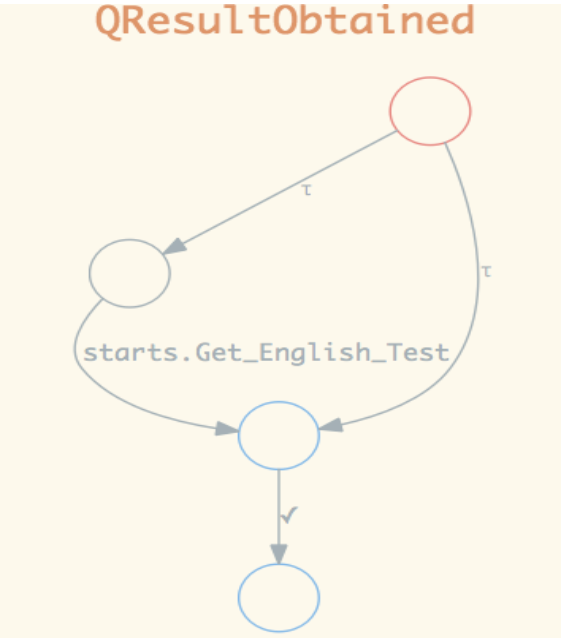
```
starts.Nominate, starts.Send_Nomination_Result,
starts.Get_Confirmation, starts.Get_Cancellation,
starts.Check_Status, starts.send_Rejection,
starts.Send_Ids )}
```

The CSP assertions hold the property specification `(e.g. QResultObtained)` on the LHS and the process specification `(e.g.UUniversityAdmission)` on the RHS and compares their traces using trace refinement `'T'`.
It hides all other tasks along with the deleted task and checks property satisfaction.

Fig 4-3 depicts the state transition system of the property `'QResultObtained'` and the university admission process after applying R1, i.e. the process without `'Get_English_Test.` Fig 4-3(a) has no task in its trace as the (RHS) assertion of `'QResultObtained'` states to hide all tasks with the removed task from the process trace. The (τ) represent an internal event (also known as tau) in CSP, which means that this event is hidden from the environment. It runs only one event which is SKIP, the successful termination. We provide a manual representation of the university admission example as FDR could not generate the LTS, see Fig 4-3(a), and this is because the process is complex and has too many states. However, this is does not affect our verification as we generate the LTS just for clarification purposes. The property `'QResultObtained'` has either the event `'Get_English_Test'` or the event SKIP on its trace, see Figure 4-3(b). Comparing the trace between the property `'QResultObtained'` and the university admission trace after hiding the tasks, as in the corresponding assertion, shows that the property and the process both has the `SKIP` on their traces. This means that the process satisfies the property and the reconfiguration here is successful. The task `'Get_English_Test'` is contributing to achieve the goal `'ResultsObtained'` but the task is not mandatory as it represents a work that is applicable for certain group of applicants. The property specification `'QResultObtained'` states that the task `'Get_English_Test'` could be deleted, see Fig 4-3(b), without any impact on the fulfilment of the process strategic goal. The trace refinement of `'QResultObtained'` and the adapted process is successful, therefore, R1 is verified to be compliant to the goal as the deletion of this task still satisfies the goal.



a) **University admission trace**



b) QResultObtained trace

Figure 4-3: Travel planning process trace

4.5 Summary

In this chapter, we have shed light on the impact of the delete policy on satisfying a business's goal. In order to measure goal satisfaction, we defined a dependency link between goals at goal level and tasks at BPMN level. This linkage allowed us to capture goal specification in property patterns and compare BPMN behaviour against these properties. From this link, we defined a goal-task dependency constraint and its satisfaction function. We also discussed how the verification of the goal-task dependency constraint is automatically performed using CSP and FDR.

Chapter 5 An Ontology-Based Approach for Insert and Replace Policies

In this chapter, we discuss how the insertion of new tasks to a workflow instance could break its original goal. Inserting new functionalities in an ad hoc manner is accomplished at the business level through the reconfiguration policy 'insert'. Due to the fact that policies introduce new functionalities which are not known prior to runtime, it is necessary to formulate the business domain and its assumptions as a fuzzy approach to be able to perform on-the-fly runtime reasoning. Domain knowledge can be expressed in ontology by expressing its concepts and their relationship in a semantic way. Domain knowledge is understood and derived from the goal in question. Therefore, capturing the requirements on ontologies enables checking the new requirements against the ontologies, which are originally derived from the goal itself. We further show how defining the domain-based ontology for BPMNs can be effectively used to validate new functionalities against their original goal.

5.1 Adaptation with Insert Policy

The insert process refers to the facility where the proposed framework allows the modification of the existing workflow by allowing the insertion of a new task into the given workflow specification. The newly inserted tasks can be of any of the following kinds of tasks:

- Atomic sequential: This refers to the insertion of a new task in sequential order immediately after a given task. This operation requires the new task name as well as the name of an existing task.
- Atomic parallel: This refers to the insertion of a new task in parallel to an existing task. The operation will insert the parallel gateway to connect the new task and existing task in parallel. The new and existing task names must be provided to perform the operation.
- Composite: The composite task is in itself a collection of multiple tasks. The framework allows the insertion of a new composite task. In this

operation, the framework will receive multiple task names that collectively represent the composite task. The framework will then insert those tasks as a composite task in reference to an existing task.

The syntax of the insert policy is defined as: `Insert (T2, T1, true)`. The translation of this syntax is insert task `T2` in parallel with task `T1`.

5.1.1 Inserting a Subprocess

The subprocess is also one of the elements that could be inserted to the BPMN. As it consists of a number of atomic tasks, its compliance to the goal can be achieved by checking each atomic task independently. The assurance mechanism tests the compliance of each task according to the domain-task conformance constraint. The subprocess satisfies the constraint only and only if all of its atomic tasks are verified to be domain conformance.

5.1.2 Inserting an Operator

As discussed in section 4.2.2, BPMN operators address the order among tasks which is outside the scope of this research. Saying that, inserting an operator can be achieved through the reconfiguration functions in the framework but we assume they are semantically correct.

5.2 Impact on Goal

All desirable actions or functionalities that any organisation wishes to achieve are basically determined through goal specification. Workflow systems capture the functional requirements in a logical order to satisfy a certain business outcome. The insert policy is used to add extra functionality to the workflow. It can insert a new workflow item (activity or operator) at

Table 5-1: Example of the insert policy and the DTC constraint

Policy	Constraint	Organisation-specific goal example	Policy example	Policy impact example
Insert	A workflow item must never be processed if does not belong to the domain	Add a new task whose goal is to offer a free delivery for VIP customer Goal: FreeDeliveryOffered Domain: PizzaDelivery	**Insert** `'Offer_Free_Delivery'`	**Desirable behaviour:** The new task is applied in the PizzaDelivery domain **Undesirable behaviour:** The new task is inserted into the university admission domain

any position. Thus, policy behaviour can be discussed from two different perspectives: what and where. The first perspective, 'what', is concerned with the new requirements to be added, while the 'where' perspective is concerned with the position or the order of existing requirements. We analyse and reason about the first perspective, which allows us to focus on the occurrence of work in workflow systems and evaluate its semantic correctness. However, order correctness and tasks compatibility have already been studied in the literature, as discussed in Chapter 2. Table 5-1 shows an example of the undesirable effect of the insert function. The constraint is general and expresses that the insertion of any undesirable action is not accepted.

In this context, semantic correctness refers to a desirable process behaviour that guarantees goal satisfaction. Current reconfiguration policies have no constraints on what to insert into the running process. Therefore, this might cause unexpected business outcomes, which in turn might affect businesses. For example, the delivery of a travel plan changes to the delivery of a washing machine.

There is a strong relationship between the goal specification and the running process, as discussed in the previous chapter. Furthermore, ontology knowledge is derived from goal specification and it is consistent with the goal in question. Hence, the satisfaction link between process and ontology lead to the satisfaction link between process and goal as the ontology is supposed to be consistent with the goal. Assume we have a Goal `(G)`, an Ontology `(O)` and a Process `(P)`, the satisfaction formulae can be written as:
`If P |= O and O |= G -> P |= G.`

Figure 1-1, in section 1.8 (Page 9), depicts the satisfaction relationship among the process, its domain and goal. The goal specification is located in the top layer since it is considered the main reference for the workflow consistency check. It is followed by domain knowledge, which represents the concepts of that domain and their interrelationship. It is consistent with the goal in question. In the bottom layer, the workflow specification, which must be consistent with the domain knowledge. The consistency between workflow and domain knowledge will lead to goal satisfaction. Hence, we use domain knowledge to reason about goal satisfaction. This is due to the fact that goal specification holds the desirable actions but is abstracted from any detail about the process for which it is designed. For this reason, we use the domain knowledge to prove consistency with the goal as it can express the wider context of the process and its original goal.
Table 5-2 shows an example of inserting a new task to the UQU admission workflow along with its syntactical definition and the reason of change.

Table 5-2: Policy example for the UQU admission workflow

Policy action description	Syntax	Reason of change
R1: insert a new task `'Re-nominate'`	`Insert (Re-nominate, Send_Ids, false)`	Applicants' satisfaction

The challenge here is how to define the desirable and undesirable actions and take the right decision at online reconfiguration. Furthermore, how to design the correct ontology and keep it updated and by whom? Despite the fact that the actions or requirements are domain specific, our constraints are generalised and could be implemented with any BPMN domain. Therefore, we decided to integrate domain knowledge with workflow systems for an engineering process, i.e., workflow verification. The unmanaged insertion of new functionalities to the running process might have a severe impact on achieving its goal.

5.3 Domain-Task Conformance (DTC) Constraint: Specification

The domain-task conformance is a goal-compliance constraint which is developed in accordance with managing the impact of the insert policy. It expresses that every task must be compliant to its domain knowledge. We define this constraint due to the nature of the insert policy and its semantic impact on achieving the goal. The effect it causes needs a fuzzy approach to allow more possibilities to take the right decision whether to allow the insertion of the new work or not. This is unlike the goal-task dependency constraint, which is defined for deleting an existing functionality. As the policy requires the deletion of an existing functionality, the goal-task dependency constraint is defined to help decide whether this functionality is goal-related or not.

Each BPMN domain is captured in an ontology in a specified way, as explained in the following section. This enables modelling the domain concepts and the semantic relationship among them. The DTC constraint ensures that any task to be inserted into the process conforms to the defined domain. We assume a process (`P`), its defined ontology, (`O`), and a new task (`T`); the task is evaluated against its domain in order to ensure its conformance. The satisfaction formulae can be written as: if `T |= O –> T |= P`.

5.3.1 Domain-Goal-Task (DGT) Ontology

In this section, we introduce an ontology for BPMNs to enrich their knowledge with extra vocabularies (i.e., tasks). There are several reasons why we choose ontology for verification purposes: (i) its natural language, which makes it easier for end users to understand, (ii) ontologies are a modelling as well as a verification language and this might save the effort and cost exploited for engineering activities, (iii) it enriches the domain knowledge as it allows for expressing related concepts to the domain along with

their semantic relations, which in turn enables online verification of self-adaptive workflows. Hence, it helps business analysts to define domain concepts represented as 'tasks' and their semantic relationship. For example, the travel planning domain is composed of travel planning concepts whose work express the semantic of this domain. It cannot include tasks whose job is to book a training course or find the nearest NHS services.

The purpose of each concept is also captured in the ontology in terms of goals, to facilitate the verification decision. Every task in the ontology must be linked to the goal that they are contributing to. Despite the fact that BPMNs are domain specific, i.e., domains differ in their goals and the purpose for which they are designed, we develop a generalised semantic constraint that could be applied to any domain. For example, Flight-Booking is a different domain from PizzaDelivery as the tasks used within the processes as well as their outcomes (goals) are different. However, the DGT ontology is a general ontology structure that can be used to integrate knowledge for any BPMN domains. Figure 5-1 depicts the DGT ontology structure.

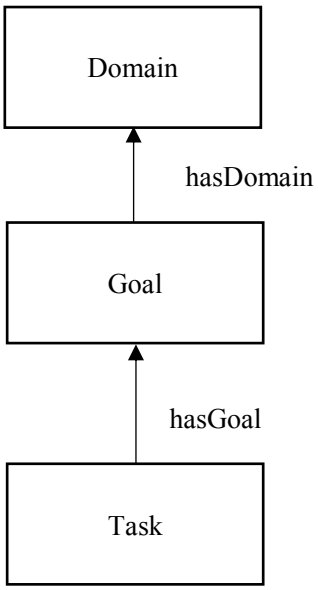


Figure 5-1: DGT ontology main construction

The DGT ontology is designed using OWL (*W3 Semantic Web, 2013*) and developed with Protégé (*Protégé, 2016*). It combines three main classes: Domain, Goal and Task. The `'Domain'` class includes the domain's name, which is unique for every domain and is

used to define a certain domain. The 'Goal' class includes goal classifications of a certain domain as expressed in the goal model. The class 'Task' encompasses all anticipated tasks in a certain domain. It is further decomposed into two subclasses: Atomic and Subprocess. The former includes tasks that are of type atomic tasks, while the latter includes composite tasks as defined as in the corresponding BPMN. Individuals of the three classes are linked using OWL object properties. Basically, we have two object properties: hasGoal, linking tasks with goals, and hasDomain, linking goals with domains.

The values for each class are defined as individuals of that class. as in the corresponding BPMN. In this work, BPMN tasks are considered to be the ontology main concepts as they are the main artefacts in process execution since they are designed to perform 'work' within the process. Furthermore, tasks are vulnerable to change at BPMN level and they are the target to be checked in the ontology.

In addition to the previous relationships, we assign the property 'sameAs' to link individuals in the class 'Task', indicating that individuals refer to the same thing in the ontology. It is defined for tasks that are doing the same sort of work to achieve a certain goal or, as we call them semantically equivalent tasks.

For example, the tasks 'Pay_by_Cash' and 'Pay_by_Card' are semantically equal as they are contributing to the same goal 'PaymentReceived'.

To illustrate the ontology hierarchy in Figure 5-3 and how it is used to express our semantical constraint, we recall our UQU admission example that introduced in Chapter 3. Its corresponding ontology is depicted in Figures 5-4, 5-5 and 5-6. The class Domain has 'UniversityAdmission' as a domain individual. All planned/unplanned tasks in the domain of the university admission are defined as individuals of the class 'Task'. They are considered as the vocabulary in that domain.

All individuals are linked to each other according to the defined semantic relationship 'hasDomain' and 'hasGoal'. The goals that are defined in the domain of the university admission are linked to their domain name under the object property 'hasDomain'. For example, the goal 'ApplicantNominated' is linked to the domain name 'UniversityAdmission' through 'hasDomain'. Hence, 'ApplicantNominated' 'hasDomain' 'UniversityAdmission'.

The tasks that belong to the ‘UniversityAdmission’ are classified under its goals through the object property ‘hasGoal’. For example, the task ‘Nominate’ is classified under the goal ‘ApplicantNominated’. Hence, ‘Nominate’ hasGoal’ ApplicantNominated’.

The key question here is how to guarantee that this new task belongs to the domain, which in turn will guarantee goal satisfaction. The ontology is built on the assumption that it encompasses all vocabularies ‘concepts’ in a specific domain. Upon this assumption, we are able to query the ontology about the availability of a specific concept ‘task’ in a specific domain. The goal-compliance framework checks the availability of the new task in the ontology and verifies its conformance with the domain only if it exists, see Figure 5-7 for the complete flowchart of the verification process. Therefore, the insertion of a new task into the process model can be automatically verified.

For example, inserting ‘deliver_Washing_Machine’ to the domain of ‘UniversityAdmission’ is semantically not accepted as it deviates from the domain semantic which is all about registering students to the right major.

Due to the fact that the business environment is dynamic and ever changing, new requirements and needs might appear. We have an assumption that the ontologies encompass everything about domains but they are also vulnerable to incompleteness due to the fact that anticipating everything at modelling time is impractical. Therefore, a more intelligent and context-based mechanism is needed to improve the decision-making process.

5.3.1.1 DGT Ontology for the UQU admission workflow

As explained in the previous section, we use the ontology to deal with the insertion or replacement of tasks. In the ontology, each task is associated with a goal through the object property ‘hasGoal’ and each goal is associated with a domain through ‘hasDomain’. We developed an ontology for the university admission domain according to the DGT ontology format. The domain is called ‘UniversityAdmission’ and it has a set of goals defined according to the goal model, depicted the previous chapter, in Figure 4-1. Every goal has a set of tasks that are defined according to their contribution to the goals along with a set of other semantically equal tasks. Figures 5-2 to 5-4 depict the university admission ontology.

The policy R1, as defined in Table 5-2, requires to insert the task `'Re-nominate'` after `'send_Ids'.` The university domain has `'Re-nominate'` as one of its vocabularies. Thus, when the framework searches the university domain, it will find this task defined and, therefore, the framework will allow the insertion through R1.

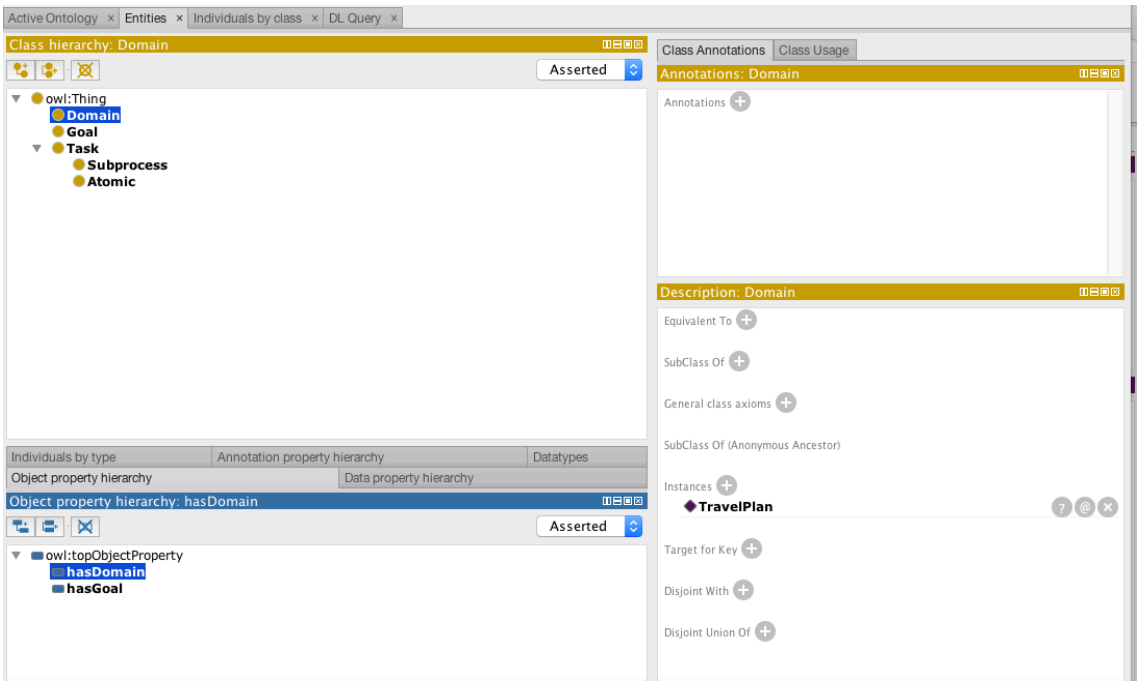


Figure 5-2: Example of the DGT ontology: classes and properties

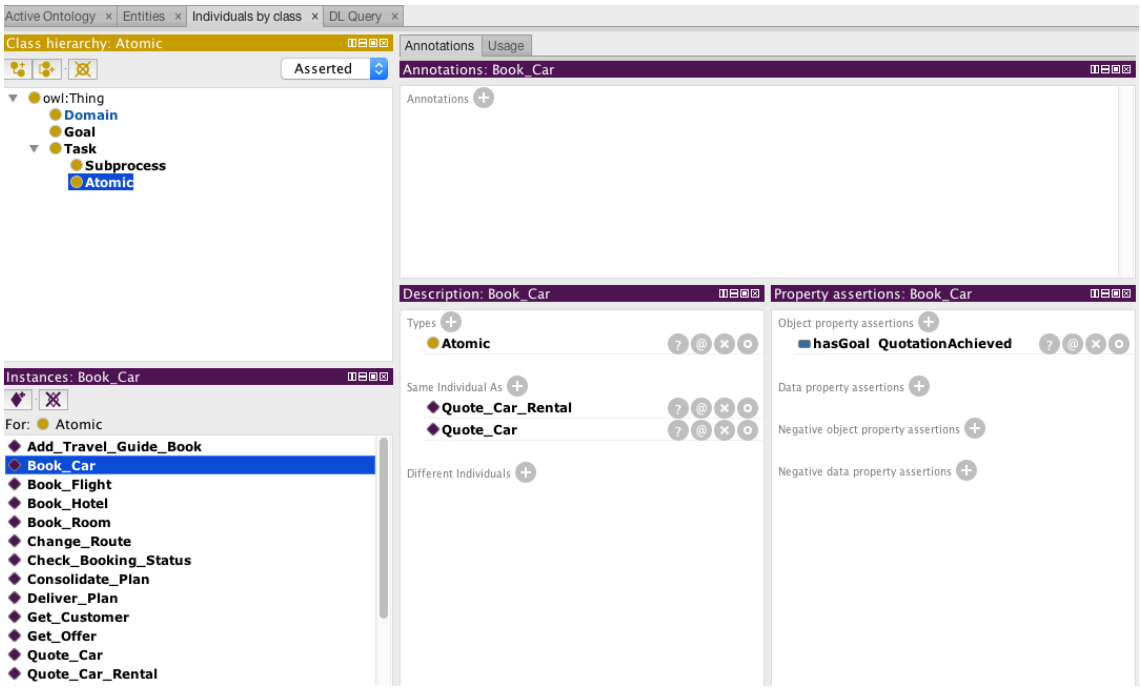


Figure 5-3: Example of the DGT ontology: Individuals

<owl:Class rdf:about="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Domain"/> <owl:Class rdf:about="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Type"/> <owl:Class rdf:about="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Task"/>	<owl:sameAs rdf:resource="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Quote_Car_Rental"/> <owl:sameAs rdf:resource="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Quote_Flight"/>
a) Classes	**b) Individuals**
<hasDomain rdf:resource="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#TravelPlan"/> <hasGoal rdf:resource="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#RequirementsChecking"/>	<owl:sameAs rdf:resource="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Choose_Ticket_Type"/> <owl:sameAs rdf:resource="http://www.semanticweb.org/budoorallehyani/ontologies/2016/11/Travel-Plan#Find_Tickets"/>
c) Object properties	**d) Individuals identity**

Figure 5-4: DGT ontology main constructs

5.3.2 The Use of WordNet

Due to the fact that it is impossible to anticipate everything about certain domains or business contexts prior to runtime (e.g., anticipating all tasks), we use WordNet to enhance the verification process. We use WordNet for searching task synonyms in case the task does not exist in the ontology. The retrieved synonyms are then matched to the existing tasks in a certain ontology. If synonyms are found, then the insertion of the new task does not violate the domain. Otherwise, the insertion of the new task is rejected and the process continues to run its original specification or stop.

A task name is assumed to be of two parts: Action and Object, separated by underscore, i.e., A_O. The first part represents an action the process is trying to perform, while the second part represents an object that belongs to a certain domain. Actions in the university admission domain include: Register, Nominate, Submit, Notify, Apply, etc. Objects in the university admission domain can be Applicant, Documents, Attainment, Test, Submission, etc.

When a policy requires the insertion of a task outside the range of the defined vocabulary in the ontology, this task is divided into action and object. Let us take the task `'Receive_Docs'` as an example where `'Receive'` is the action and `'Docs'` is the object. Thus, it could be broken down into `'Receive'` and `'Docs'` when using WordNet to find their synonyms. The retrieved synonyms are then checked against the ontology where both must be matched with the defined tasks in the ontology. The action part is related to the goals in the 'Goal' class which ensures to which goal the new task belongs. For example, one of the retrieved synonyms from WordNet for `'Receive'` is `'Get'` which is already exist in the ontology and linked to the goal `'InformationRetrieved'` and one of the retrieved synonyms of `'Docs'` is `'Docs'` in which it already exists as well. Therefore, the task `'Receive_Docs'`conforms to the university admission domain and can be inserted to the running process.

5.4 DTC Algorithm: Verification

The procedure developed for the insertion of the new task into an existing workflow is the same irrespective of the above-mentioned types in Section 5.2. This procedure can be

seen from the detailed flowchart presented in Figure 5-5. A short explanation is provided below:

1. In the first step, the framework reads the existing workflow specification and then obtains the new task name from the configuration file that contains the modification details. This name is then searched from the available ontology. This search query targets that the task name must match an individual name in the ontology, satisfying the constraint that the individual must belong to the same domain as the domain of the workflow specification. If the search succeeds, then the Re-configurator will allow the insertion of the new task. Otherwise, it will carry out Step-2.
2. In case the given task name is not available in the ontology, then the framework will attempt to explore the possibility of confirming the suitability of that task name through WordNet. The framework assumes that the task name must consist of two words separated by a special character (e.g. "_"). The first word represents the action, where the second word represents the object (e.g. "Register_Student"). The framework interacts with the WordNet repository to obtain the synonyms of both words (i.e. object and action parts). The object part and its synonyms help to identify the corresponding domain of the workflow, whereas the action part hints at the type of action.
3. Once the synonyms are retrieved, then they are searched in the ontology. The framework will allow the insertion of the new task, if any of the synonyms of both parts are found in the ontology. Otherwise, the framework will not allow the insertion of the new task.

It is integrated with the framework and it is compiled using OWL API (*OWL API, 2015*) for Java. This library basically provides interfacing with OWL API to interact with the Ontology file. This class contains all the necessary operations that are needed for verification. The compilation is activated when a policy requires the insertion of a new task for verification purposes. The WordNet API (*WordNet, 2012*) is also integrated within the framework for verification purposes as discussed earlier.

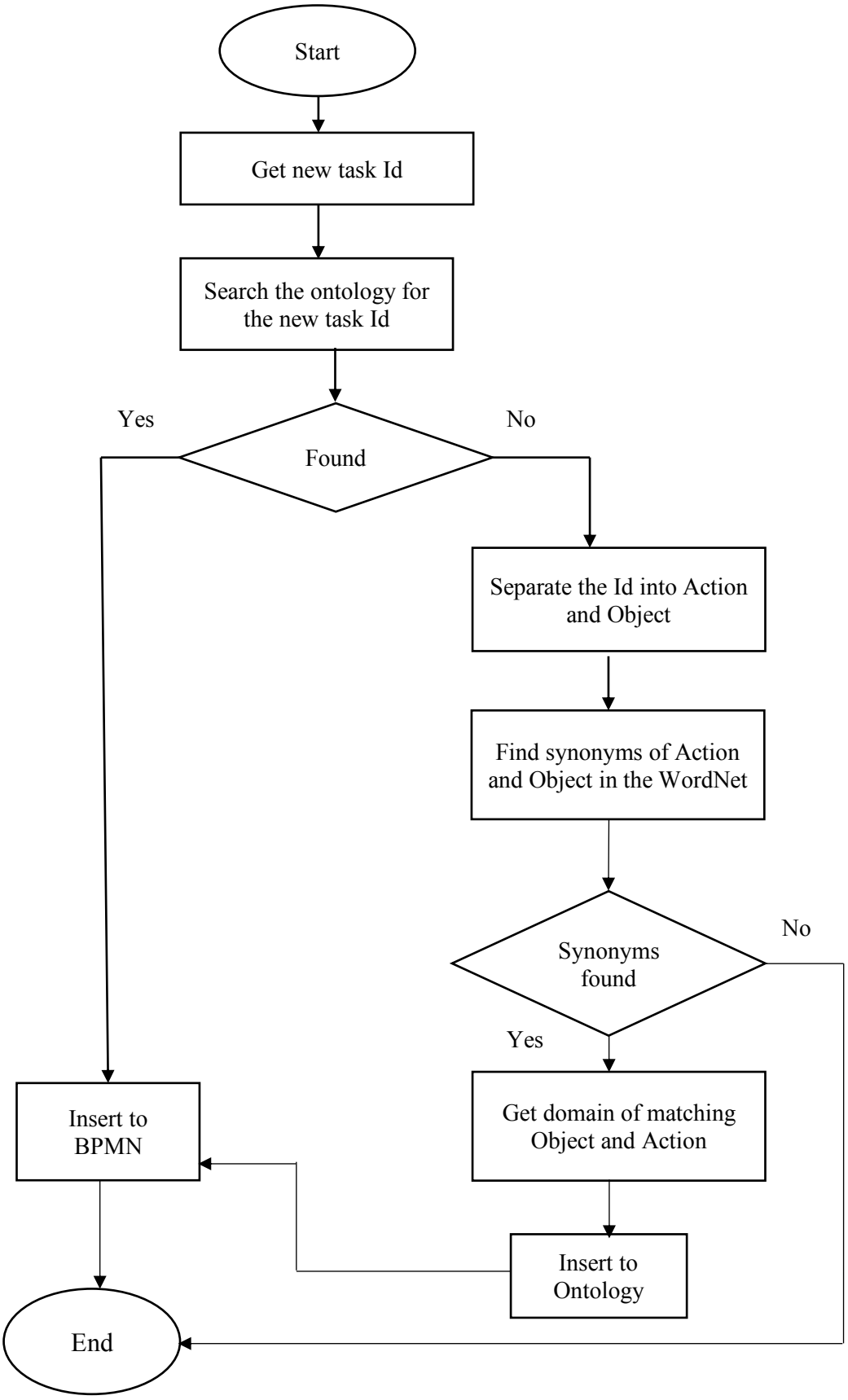


Figure 5-5: Flowchart of DTC verification mechanism

5.5 Adaptation with Replace Policy

In addition to the reconfiguration functions defined in (Gorton, 2011), we introduce the replace policy as a variant of reconfiguration functions. Replace is a complex change function as it consists of two different functions: delete and insert, in either order. Furthermore, it should be synchronised with the instance until it performs the complex change correctly. For example, a policy performs replacement of task `'Pay_by_Card'` with `'Pay_by_Cash'`. There are two different possibilities to achieve that: (a) it deletes task `'Pay_by_Card'` first and then inserts `'Pay_by_Cash'`, or (b) it inserts task `'Pay_by_Cash'` in the process and then deletes `'Pay_by_Card'`. In both cases, the framework allows the replacement if it does not violate the specified goal. Similar to the insert policy, the framework enables the replacement of atomic tasks and subprocesses either sequentially or in parallel.

5.6 Impact on Goal

The impact of both cases on goal satisfaction is basically the same as the impact of delete and insert policies. As a result, the framework follows goal-compliance constraints to validate the replacement. However, the framework has two different cases for verification when considering the aforementioned cases independently.

The first case (a), the framework validates the change according to the goal-task dependency constraint as it first deletes an existing task and, based on the verification result, it decides to accept or refuse the change. If the constraint is successfully validated, the framework deletes that task and performs the insertion afterwards. Before it inserts the new task, it checks its validity through the domain-task conformance constraint. If it is validated, then the framework inserts it. Otherwise, it is neglected and runs the original process but with a missing task. This is an unsafe situation (weak behaviour) as although this removed task is meant to be removed from the process, a new task is supposed to take its place. This might affect the original goal and result in undesirable behaviour.

The second case (b), however, it is the same effect but slightly different in the sense of redundancy. The policy requires the insertion of a new task first, so its validity is checked. If it is confirmed, a new task is inserted and then an existing task should be removed. However, if this task is a goal-related task, the framework will not allow this policy to be

applied. Therefore, this task might be redundant as it is meant to be replaced with another task or it might affect the behaviour in an undesired way. Hence, both cases ae too restrictive and not acceptable in practice, while the replacement might be of great advantage to the running instance. Therefore, we develop a different constraint to deal with this type of change instead of implementing the same constraints of delete and insert functions.

Table 5-3 shows an example of the undesirable effect of the replace function. The constraint is general and expresses that the replacement with any undesirable action is not accepted.

5.7 Task-Task Consistency (TTC) Constraint: Specification

The constraint task-task consistency can be seen as a variant to the domain-task conformance constraint because the replacement always means inserting a new task. The TTC constraint expresses that the new task must always be consistent with the task to be replaced with in order to guarantee goal satisfaction. The consistency here determines that the new task must contribute to the achievement of the same goal as the replaced one. There are two possibilities to express this semantic property within the ontology as the new task either exists in the ontology or does not.

The existing tasks in the ontology can be defined as `'sameAs'` if they are semantically equal. Furthermore, the property `'hasGoal'` link tasks that are contributing to achieve the same goal. The declaration of these properties facilitates to express the consistency among tasks. The TTC constraint is defined based on this consistency and it expresses that any new task must be consistent with the task to be replaced with. If both exist in the ontology, both must belong to the same goal through `'hasGoal'` or both tasks are defined in the ontology as `'SameAs'` individuals. For example, the tasks `'Pay_Tickets_By_Cash'` and `'Pay_Tickets_By_Card'` are defined as being semantically equal as they hold the same 'work' and contribute to the same goal, which is 'TicketsPaid'. Thus, if one of those tasks replaces the other it does not violate the domain semantics. Otherwise, if the new task does not exist in the ontology the framework retrieves its synonyms from WordNet and checks their conformance against the ontology.

Table 5-3: Example of replace policy impact and the TTC constraint

Policy	Constraint	Organisation-specific example	Policy Example	Policy impact example
Replace	A workflow item must be replaced by a semantically equal item or an item that is contributing to the same goal	A customer must be notified when his delivery is on its way	**Replace** `'Email_Notification'` With `'SMS_Notification'`	**Desired behaviour:** the customer will be notified by SMS **Undesired behaviour:** the customer will not receive a notification and the goal will not be achieved

5.8 TCC Verification

The replace process of the proposed framework allows the replacement of an existing task with a new one. The existing task in the process that is meant to be replaced with a new task already exists in its corresponding ontology. When the framework deletes this task from the process and inserts the new task instead, it follows the steps below (see Figure 5-6):

1. It looks up the ontology for the existing task and retrieves its synonyms through the semantic relation `'sameAs'`, if any
2. It then compares the new task with the retrieved synonyms
3. If it matches any of them, it successfully satisfies the TTC constraint
4. Otherwise, the framework checks if both tasks exist in the ontology and belong to the same goal in the hierarchy through `'hasGoal'`
5. If they belong to the same goal, the replacement successfully satisfies the TTC constraint

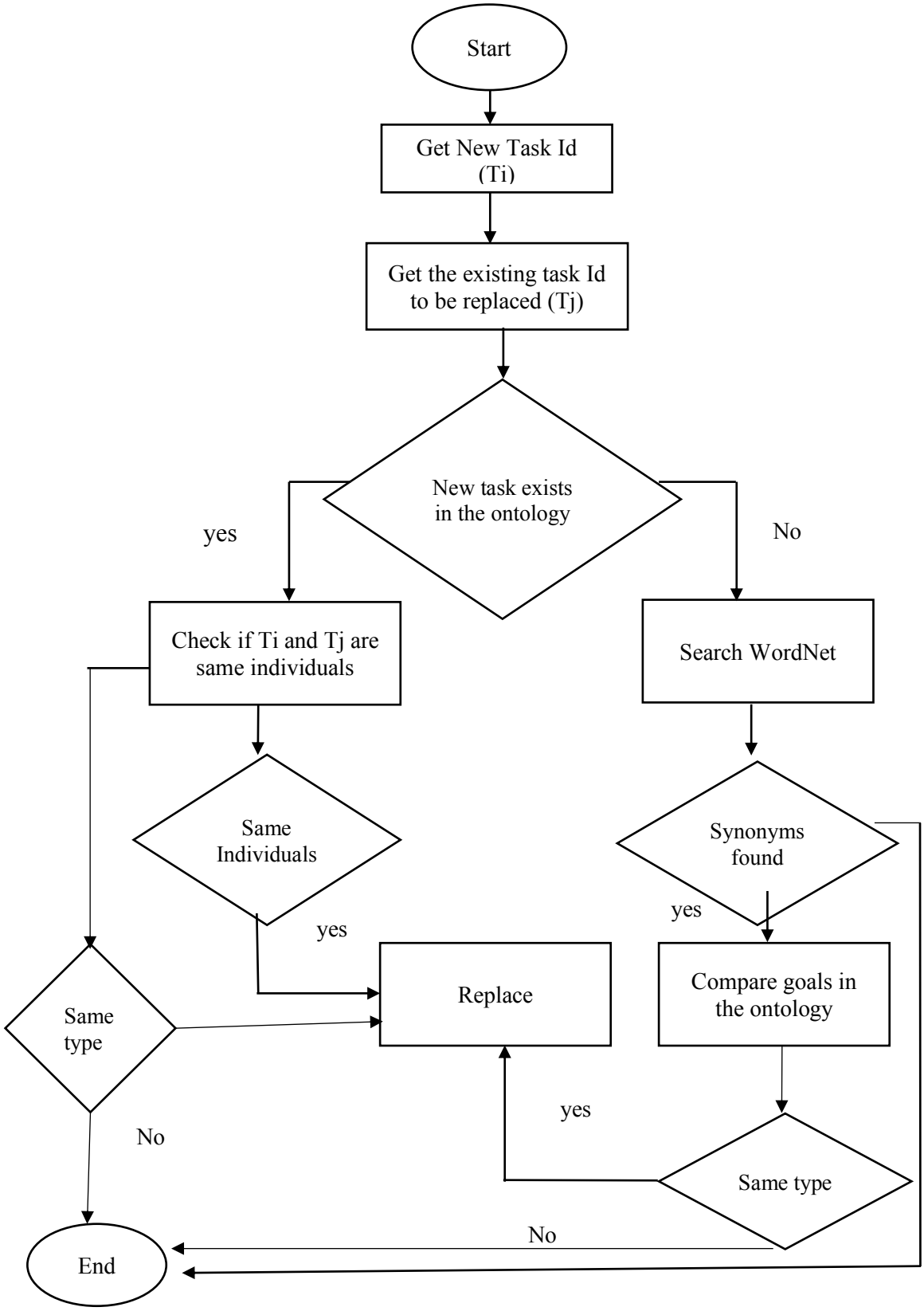


Figure 5-6: Flowchart of TCC verification

6. Otherwise, the framework retrieves the synonyms of the new task from WordNet
7. If synonyms found in the domain, the replacement successfully satisfies the TTC constraint
8. Otherwise, the framework rejects the replacement and runs the original specification instead or generates feedback to the user for human decision.

Recalling back our university admission example, assume a policy requires to replace the task `'SMS_Notification_Of_Submission'` with `'Email_Notification_Of_Submission'`. As the policy is of type 'replace', the framework will check the satisfaction of the TTC constraint. The framework checks the ontology if both tasks are defined to be the same individual through the property `'sameAs'` or they are related to the same goal through the property `'hasGoal'`. If one of these constraints hold, the verification is successful and this means that the policy is applicable. In this case, both tasks are defined in the university admission ontology as same individuals, therefore, this policy successfully satisfies the constraint and the framework allows the replacement.

5.9 Summary

In this chapter, we have analysed and discussed the impact of inserting new functionalities and replacing tasks at instance level. We also have defined goal-compliance constraints to ensure compliance to the original goal during runtime adaptation. These constraints are domain-task constraint and task-task consistency constraint. Each constraint is implemented within the goal-compliance framework to control policy behaviour.

Chapter 6 Evaluation

6.1 Introduction

The proposed framework provides assurances on goal satisfaction in self-adaptive workflows. It performs instance adaptation in response to changing requirements and supports reasoning about the adaptation to ensure new requirements are compliant to the original goal. The proposed framework is supposed to be embedded into the runtime environment, where the workflow and the policy engines are established. The workflow and policy engines support and manage the functionalities related to the adaptation, while the functionalities related to assurances are supported and managed with the proposed goal-compliance mechanisms.

In section 1.3, our hypothesis presume that goal compliance assurance can be reliably realised by existing verification techniques. Existing verification techniques are stable and it is argued (Tamura *et al*, 2013) that this stability does not meet the dynamicity nature of self-adaptive systems. In this research, we used existing verification techniques (trace refinement and domain-knowledge analysis) and we show they can be used for self-adaptive workflows with minimal user intervention. In section 6.6 we discuss to what extent these techniques helped us to achieve our aim.

For applicability and reliability evaluation, we assess the proposed approach considering several angles: framework performance, framework adequacy and ontology accuracy. The applicability is concerned with the framework performance and its ability to meet the runtime desirable criteria while the reliability is concerned with its ability to balance the need for adaptation and the quality over adaptation.

In the following section, we discuss the methodology used for achieving our evaluation. Section 6.3 shows the conducted experiments and their results for performance and accuracy evaluation. Section 6.4 discusses the framework adequacy using workflow patterns as the benchmark. We also discuss the capabilities and limitations of the proposed approach regarding the desirable runtime criteria in section 6.5.

6.2 Methodology

As our evaluation targets several perspectives as explained in the previous section. For evaluation purposes we used two different methodologies; Java code and runtime criteria.

A. Java code

We developed a Java code for evaluating the framework performance. The code is simple and basically calculates the time taken by the framework to execute its functions. These functions, as introduced in Chapter 3, are responsible for accomplishing the main functionalities of the proposed framework. They are varied based on the reconfiguration type as discussed in section 6.3. The execution time is recorded for each function. The program records the start time when the function is called and start its work as well as the time when the function completes its work. The overall time for each function is calculated by subtracting the start time from the end time. The framework records the time for all functions in an excel file that is given as an input in the appConfiguration file. Section 6.3.1 discusses the framework performance evaluation in terms of the time taken to perform its functionalities.

B. Ontology evaluation criteria

The framework accuracy is evaluated considering the ontology accuracy criteria (Hlomani and Stacey, 2014). We conduct experiments with various BPMNs' complexity and domain. Ontology evaluation is categorised under two different perspectives, including ontology correctness and ontology quality (Hlomani and Stacey, 2014). Each of which has different metrics for evaluation. In this thesis, we consider evaluating the developed ontology according to its accuracy.

The accuracy is classified as a correctness metric and it includes precision, recall and coverage as the main criteria. However, we are going to evaluate the DGT ontology targeting only the precision and recall as we aim to show how effective the ontology can be for runtime verification. While the coverage is not our focus for evaluating the developed ontology as we assume the ontology cover everything about a specific domain. Precision and recall are well-known criteria in the information retrieval filed. Precision is defined as the total number correctly found over whole knowledge defined in the

ontology, whereas recall measures the total correctly found over all knowledge that should be found. Section 6.3.2 shows the conducted experiments along with their results.

C. Workflow patterns

The framework adequacy is evaluated in terms of its ability to verify a wide range of the workflow patterns. Workflow patterns are defined in (Russell, 2006) for the purpose of indicating the requirements of the workflow languages. They are 43 patterns in total and classified to several groups; basic control-flow patterns, advanced branching and synchronisation patterns, structural patterns, multiple instance patterns, state-based patterns and cancellation patterns.

Gorton (Gorton, 2011) provides an evaluation on the reconfiguration policies considering these patterns and shows that the reconfiguration policies directly support 19 patterns and indirectly support 10 other patterns, making a total of 29 out of 43. Our approach is built upon Gorton's work with the focus on the verification side of the reconfiguration policies. So, we provide an assessment of the proposed framework against these patterns in section 6.4.

6.3 Experiments

In the previous section we introduce the methodology we used for evaluation purposes. The framework performance and accuracy are evaluated by conducting experiments as will be discussed in the following subsections.

6.3.1 Framework Performance

Although the performance was not our aim when developing the framework, measuring its performance is an essential requirement for runtime verification. We give an initial envision for its applicability in practice. The performance aims to measure the execution time taken by the framework to perform its aforementioned functions. We consider two main factors for the framework performance evaluation: (1) workflow complexity and (2) reconfiguration complexity. The former is related to the workflow size in terms of the number of elements it contains, mainly tasks. The latter refers to the number and type of the reconfiguration functions per a single instance.

The experiments were conducted with randomly and automatically generated BPMN models of various size. Thus, we consider different sizes of workflows, from simple

workflow containing 5 tasks and two gateways to very complex workflows with 50 and 100 tasks with random number of gateways.

We also perform complex reconfiguration by implementing multiple reconfiguration functions at a time. For example, inserting a new task and deleting an existing one at the same time. Please note that when the framework performs multiple reconfigurations of different types (insert and delete), it performs two different types of verification at the same time (Ontology checking and FDR trace refinement). We believe that real world processes are long running transactions and the change might be of a complex nature per running instances.

As we discussed earlier, the verification process differs according to the reconfiguration type. The framework records the time taken to perform reconfiguration and verification functions. Table 6-1 shows the functions associated with each reconfiguration type.

Table 6-1: Measurement functions per reconfiguration

Reconfiguration type	Recorded functions
Delete	• Reading appConfiguration file • Reconfiguration • CSP conversion • FDR verification • Overall time calculation
Insert	• Reading appConfiguration file • Reconfiguration • Ontology check • WordNet check (if needed) • Overall time calculation

In the following two sections, we discuss the conducted experiments along with the results for performance evaluation, according to the workflow complexity and reconfiguration complexity.

6.3.2 Impact of Workflow size on Framework Performance

The goal-compliance framework provides three types of compliance check as illustrated in Chapters 4 and 5. Its main aim is to ensure goal-compliance during runtime reconfiguration. The verification of deleting workflow tasks is accomplished in cooperation with FDR, while the latter is accomplished with ontology and/or WordNet within Java-Protégé and Java-WordNet collaborations.

6.3.2.1 Experiments

We run test experiments by randomly increasing the workflow size. The framework automatically generates different sizes of workflows by increasing the number of tasks. The tasks are generated following the last task from the provided BPMN and giving name ids as TestTask_i, where i ranges from 0 to N-1 (where N is the target number of tasks). For example, to increase a giving BPMN with 100 tasks, the framework starts inserting TestTask_0 after the last existing task and continues to TestTask_99.

Table 6-2 shows the reconfiguration type, the workflow size and the time taken to execute and validate reconfigurations for each experiment. The experiments are conducted using a workflow that initially consists of 10 atomic tasks and an ontology file of 70 task individuals.

The reconfigurations are changed per experiments, i.e., E1 with 10 tasks performs simple reconfiguration (single change) and complex reconfiguration (multiple change at a time).

Table 6-2: The execution time according to workflow complexity

Experiment Id	Reconfigurations	Workflow Size	Overall average time
E1	• Insert an atomic task after TestTask_5	10	1.32s
E1	• Delete an atomic task (TestTask_7) • Delete an atomic task (TestTask_8)	10	1.48s
E2	• Insert an atomic task after TestTask_47	50	1.60s
E2	• Delete an atomic task (TestTask_44) • Delete an atomic task (TestTask_38)	50	1.91s
E3	• Insert an atomic task after TestTask_88	100	1.80s
E3	• Delete an atomic task (TestTask_63) • Delete an atomic task (TestTask_92)	100	2.32s

The assertion file consists of the goal properties specification and their corresponding assertions for calculation through the FDR.

The framework is provided with three appConfiguration files containing information about each experiment where every experiment is assigned to a different file. However,

we provided the number of targeted runs in the main class in order to run them all at the same time. We run them two times and calculated the average overall time as shown in Table 6-2.

Figure 6-1 shows the relationship between BPMN size and the execution time taken by the framework to perform a simple reconfiguration. Whereas, Figure 6-2 shows the time taken to perform a complex reconfiguration as increasing the BPMNs complexity. We noticed that the average execution time grows slightly linearly with the workflow size as well as with the reconfiguration policies. Thus, scalability is not a problem as the framework still behaves well when the workflow size is increased. However, it is believed that there is a positive correlation between BPMN size and errors (Mendling, Reijers and van der Aalst, Wil MP, 2010) and the size is suggested not to be more than 50 elements.

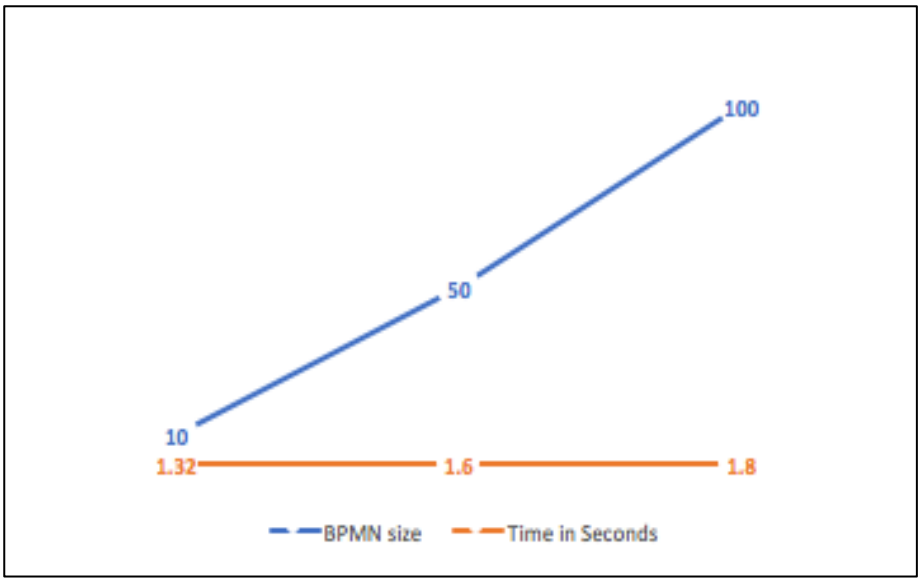


Figure 6-1: Correlation between time and BPMN complexity with single reconfiguration

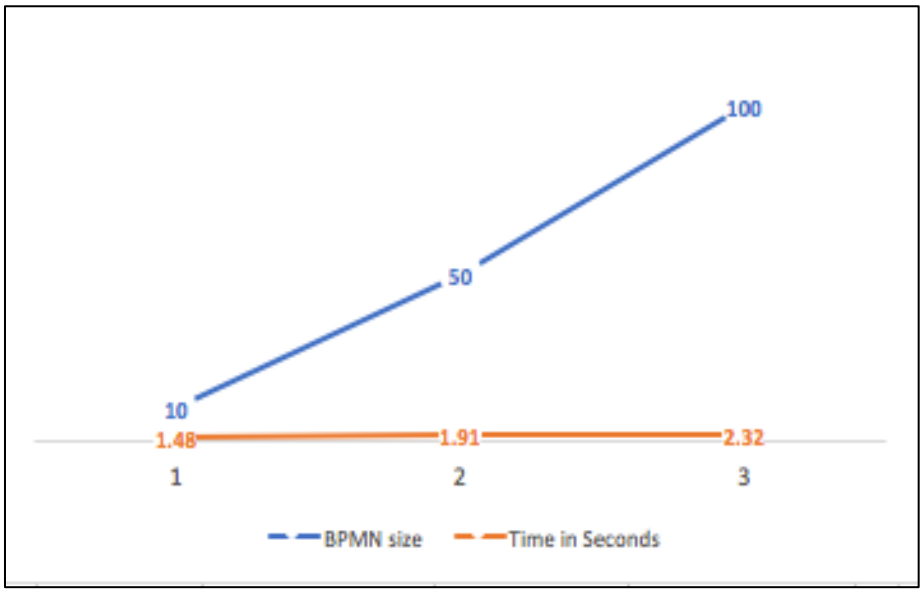


Figure 6-2: Correlation between time and BPMN complexity with complex reconfiguration

We also measure the time taken to perform reconfiguration and verification on complex BPMNs by changing the position of inserting new tasks or deleting existing ones. The position is changed from inserting/deleting to/from the beginning/middle/end of the BPMN. The results showed insignificant change in the execution time either the change was at the beginning/middle or the end of the BPMN.

As the framework uses FDR to perform the verification that is related with the delete policy, we evaluate its performance independently from the framework. FDR showed that the average time to calculate a simple assertion that checks the availability of a single task, e.g. the task `'Confirm_Booking'` is contributing to achieve the objective 'FlightBooked' in the Travel domain, is 0.81s. While some assertions can be more complex, i.e., the assertions that check the availability of group of tasks. For example, the tasks `'Quote_Flight'` OR `'Quote_ Hotel'` OR `'Quote_Car'` are contributing to achieve the objective 'QuotationAchieved'. For complex properties, the average time taken to calculate the assertion is 0.2s.

6.3.3 Impact of Complex Reconfiguration on Framework Performance

In this section, we discuss the impact of performing complex reconfigurations on the framework's performance, the execution time in particular. Complex reconfiguration is the second perspective in our performance evaluation. It refers to the number of reconfiguration functions the framework can perform per instance. However, they are

implemented concurrently at the same time per a single instance. The developed appConfiguration file gives the ability to add as many reconfiguration operations as we wish, as explained in Chapter 3. Each time the framework was requested to make complex changes to the running BPMN, it checks the verification of each change at the same time. Table 6-3 shows the experiments along with the changing complexity in the type and the number of the reconfigurations with the average time taken by the framework.

Table 6-3: Summary of experiments for evaluating performance according to reconfiguration complexity

Experiment	Reconfigurations	Average time
E1	• Delete an atomic task • Delete an atomic task (the task is a branch of an XOR gateway)	1.41s
E2	• Insert an atomic task (the task exists in the ontology) • Insert an atomic task (the task does not exist in the ontology)	1.12s
E3	• Delete an atomic task • Insert a composite task in parallel • Insert an atomic task (the task to be inserted as a new branch to an XOR gateway)	1.49s
E4	• Delete a composite task • Insert an atomic task • Insert an atomic task in parallel with the previous one	1.49s

6.3.3.1 Experiments

We consider measuring the performance of goal-compliance framework when performing complex reconfigurations. Three experiments were conducted using the same workflow with three different reconfiguration functions. The workflow consists of 5

atomic tasks and 2 parallel gateways with single start and end events. The reconfiguration functions are changed per experiment as explained below. While the reconfiguration varies according to several aspects including (1) the number of reconfigurations at a time, (2) the type of the reconfigurations (insert/delete) per experiments, (3) the nature of the reconfiguration function (sequence/parallel) and (4) the degree of complexity of the task being inserted or deleted (composite/atomic task).

Figures 6-3 and 6-4 show the time taken to perform complex reconfiguration of type 'delete' and 'insert', respectively. Please not these charts show the time taken per a single run per each experiment not the average time. The execution time taken by the framework to perform complex deletion is slightly more than the time taken to perform complex insertion. The reason of this might be because of the FDR invocation and calculation within the framework when deleting tasks, whereas with the insertion the framework just interface with the ontology and WordNet through the Java libraries.

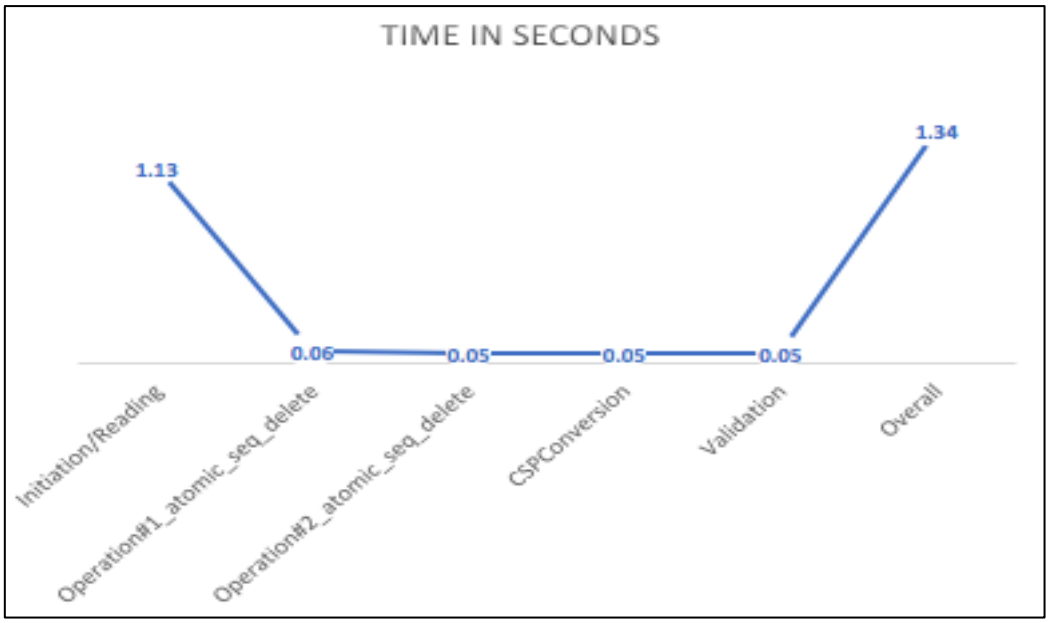


Figure 6-3: Time taken to perform complex reconfiguration of type 'delete', E1

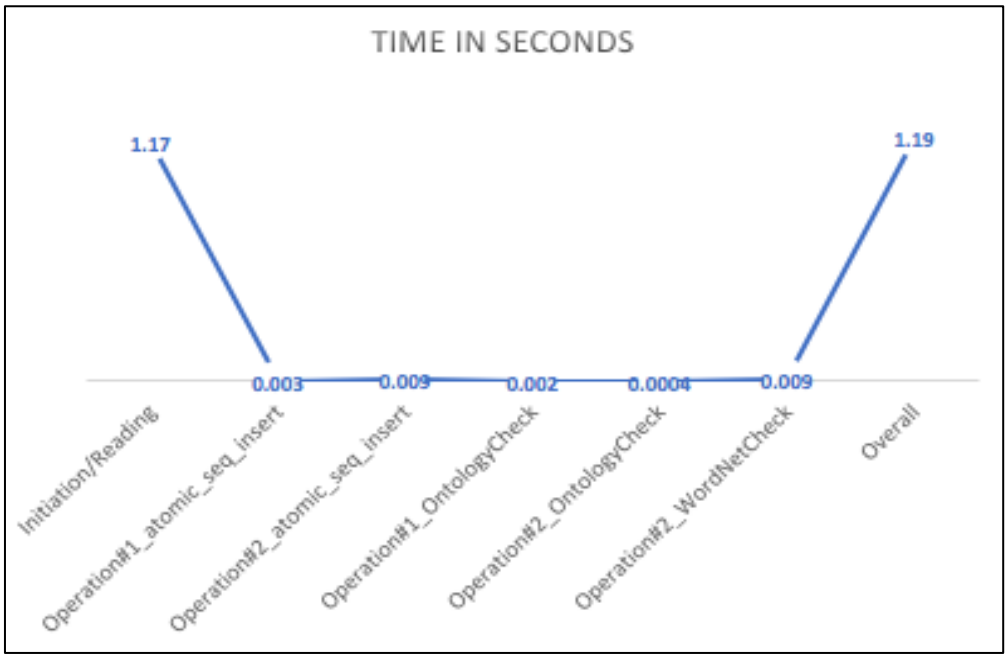


Figure 6-4: Time taken to perform complex reconfiguration of type 'insert', E2

Figures 6-5 and 6-6 show the time taken to perform complex reconfiguration of different types. Please note these charts show the time taken per a single run per each experiment not the average time. The experiments conducted by changing the other aspects, i.e. nature of reconfiguration and complexity of tasks. The average time of both experiments E3 and E4 show that the nature of reconfiguration and complexity of tasks do not have a significant impact on the framework performance.

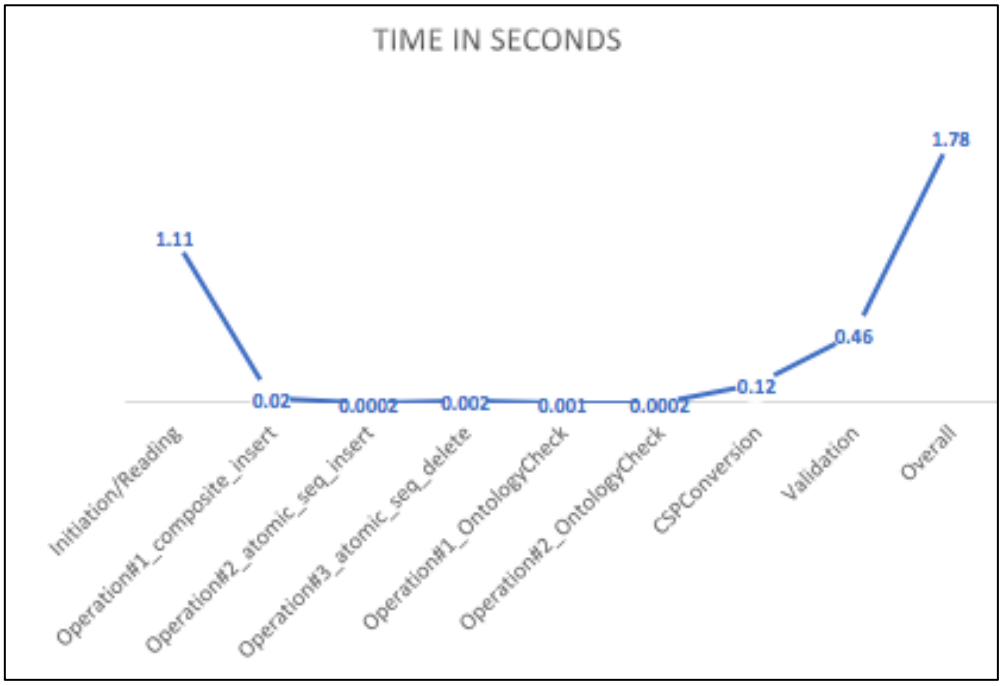


Figure 6-5: Time taken to perform various complex reconfigurations, E3

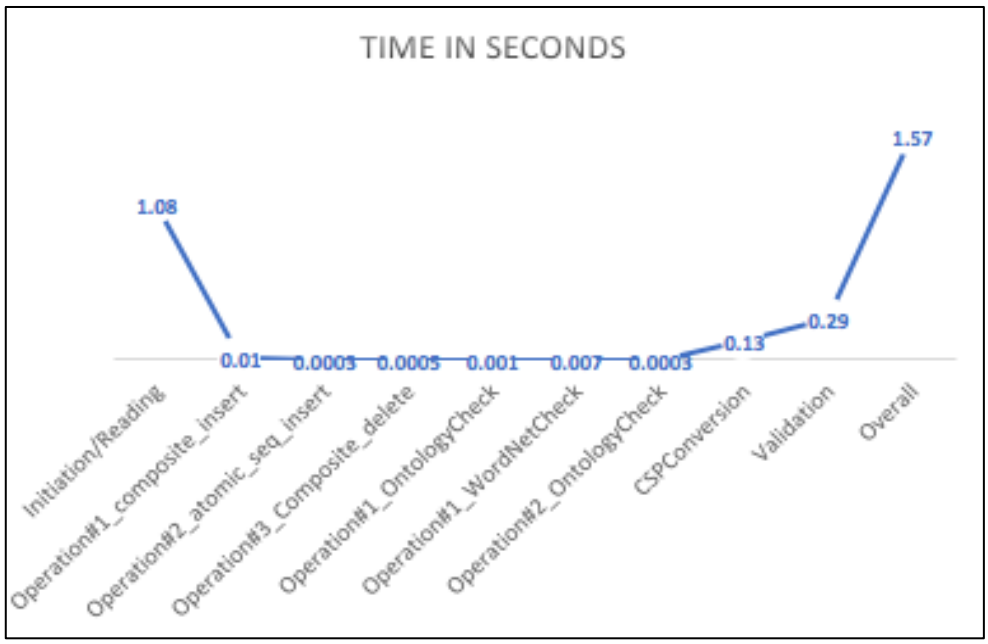


Figure 6-6: Time taken to perform various complex reconfigurations, E4

6.3.4 The DGT Ontology Evaluation: Ontology Accuracy

The proposed DGT ontology can be generalised to represent any BPMN domain. It is based on an assumption that it encompasses all tasks (designed and un-designed) that belong to a specific domain. However, predicting all tasks related to instance variants is impossible at modelling time. As a result, WordNet was integrated within the framework to enhance the verification process.

Ontology is expressed in a natural language, which makes it easier for an end-user to understand. Ontology evaluation is categorised under two different perspectives, including ontology correctness and ontology quality (Hlomani and Stacey, 2014). Each of which has different metrics for evaluation. In this thesis, we consider evaluating the developed ontology according to its accuracy. The accuracy is classified as a correctness metric and it includes precision, recall and coverage as the main criteria. However, we are going to evaluate the DGT ontology targeting only the precision and recall as we aim to show how effective the ontology can be for runtime verification. While the coverage is not our focus for evaluating the developed ontology as we assume the ontology cover everything about a specific domain. Precision and recall are well-known criteria in the information retrieval filed. Precision is defined as the total number correctly found over whole knowledge defined in the ontology, whereas recall measures the total correctly found over all knowledge that should be found.

The experiments implemented on three different BPMN models from different domains; PizzaDelivery, CabBooking and TravelBooking. The BPMN models vary on the number of tasks (from 7 to 12 tasks). We developed an ontology for those BPMNs, each one contains different number of task individuals as shown in Table 6-4. Ontologies are designed and integrated with the proposed framework. We run the experiments by applying the insert function which reads the new task id, the nature of the insertion (sequence/parallel) and the input/output files. Each time the policy requires to insert a new task, the framework validates the request before allowing to proceed with the insertion (this point was discussed earlier in Chapter 5). The number of verified tasks matched the tasks defined in the class `'Task'` in the ontology, was 54 out of 60. Whereas six tasks were not found in the ontology directly, but four of them were matched through synonyms finding with WordNet, making a total of 58. However, two tasks were failed to meet the DTC constraints and as a result were rejected.

Table 6-4: Experiments for ontology evaluation

BPMN domain	Number of tasks in the ontology (whole knowledge)	Number of experimented tasks	Number of matched tasks
Pizza delivery	57	20	18
Cab booking	44	20	18
Travel booking	37	20	18

The average of ontology precision according to the experiments, as shown in Table 6-4, is 40%. We think the ontology precision is promising and the resulted percentage is 40% because the number of the experimented tasks is small compared to the whole knowledge. The average of ontology recall according to the same experiments is 90%.

There are several reasons why the framework failed to validate some tasks against the ontology and WordNet. The first reason is that framework only handles task ids in the Action-Object format. It retrieves their synonyms through WordNet and finds them in the ontology. If the new task id follows the Action-Object-Object format, the framework can handle it if it exists in the ontology but cannot retrieve its synonyms from WordNet. The other reason is that WordNet does not have all synonyms as we expect. For example, task `'Hire_Car'` failed to be inserted in the TravelPlanning domain, while the ontology has vocabulary such as `'Book'` and `'Find'` but they are not defined in the WordNet as synonyms to `'Hire'`.

In case of long task id, the framework should determine the most relevant object with the domain in question. Failure to assign the most relevant object might result in failure to decide the task conformance within the given domain. Furthermore, the framework should be enhanced with more intelligent techniques for matching and decision making.

6.4 Framework Adequacy: Workflow Patterns

The goal-compliance framework is supposed to execute BPMN processes and reconfigure their structure with StPowla reconfiguration functions. BPMN processes represent workflow systems and StPowla targets the reconfiguration at workflow systems. As our approach assumes workflows to be modelled by BPMN notations and their reconfigurations is realised by StPowa policies, we consider the workflow patterns for evaluating the adequacy (coverage) of our framework. The workflow patterns are

generalised for workflow systems regardless the modelling notation (e.g., BPMN, XPDL, Petri Nets, etc) and they are widely acceptable. Table 6.5 shows the patterns and the supported patterns by BPMN and StPowla and highlights the tested patterns by the goal-compliance framework in addition to the supported but not yet tested patterns.

In Table 6-5, the green highlighted cells refer to supported/tested patterns, the red highlighted cells refer to the unsupported/untested patterns, the yellow highlighted cells refer to the partially supported patterns and the blue cells refer to the supported but not yet tested patterns. The supported but not tested patterns are those patterns which are supported (including the partially supported patterns) by BPMN and StPowla but not yet tested by our framework. The unsupported patterns by the goal-compliance framework are not supported by both BPMN and StPowla. However, regardless of the workflow specification language and the adaptation logic, we think the proposed verification algorithms could be generalized to verify compliant to the goal in self-adaptive systems.

Table 6-5: Evaluation of goal-compliance framework in relation to workflow patterns

Patterns	BPMN	STPOWLA	Tested	Supported but not tested
Sequence				
Parallel Split				
Synchronisation				
Exclusive Choice				
Flow Merge				
Multi-Choice				
Structured Synchronising Merge				
Multi-Merge				
Structured Discriminator				
Blocking Discriminator				
Cancelling Discriminator				
Structured Partial Join				
Blocking Partial Join				
Cancelling Partial Join				
Generalised AND-Join				
Local Synchronising Merge				
General Synchronising Merge				
Thread Merge				
Thread Split				
Multiple Instances without Synchronisation				
Multiple Instances with a Priori Design-Time Knowledge				
Multiple Instances with a Priori Run-Time Knowledge				
Multiple Instances without a Priori Run-Time Knowledge				
Static Partial Join for Multiple Instances				
Cancelling Partial Join for Multiple Instances				
Dynamic Partial Join for Multiple Instances				
Deferred Choice				
Interleaved Parallel Routing				
Milestone				
Critical Section				
Interleaved Routing				
Cancel Task				
Cancel Case				
Cancel Region				
Cancel Multiple Instance Activity				
Complete Multiple Instance Activity				

Arbitrary Cycles				
Structured Loop				
Recursion				
Implicit Termination				
Explicit Termination				
Transient Trigger				
Persistent Trigger				

6.5 Discussions

This section discusses the capabilities and limitations of the proposed framework in general and its applicability according to the desirable runtime verification. Although the framework guarantees the goal-compliance properties over adaptive BPMNs in most cases, it fails at some cases (in particular the cases related to the insertion) for several reasons. First of all, due to the fact that ontology is incomplete as it is impossible to predict every related vocabulary at design time. However, we suggested the use of WordNet to overcome ontology incompleteness. Second, WordNet dictionary does not include all the expected synonyms. Finally, the defined constraints are restrictive to some extent as we believe they need to be enhanced with intelligent techniques to offer a better degree of reliable flexibility. Therefore, this leads to restrictive adaptation through the proposed framework. However, the framework is characterised by atomicity, which is a desirable feature in runtime verification (Villegas *et al*, 2011). Atomicity means that the framework has the ability to run the original specification in case the adaptation fails. It also generates a feedback to the authorised users for a human-assisted decision.

Self-adaptive systems adapt their structure, without human intervention in the midst of their executions. Therefore, validation and verification mechanisms for such systems must be characterized by runtime V&V properties including: sensitivity, isolation, incrementality and composability (De Lemos *et al*, 2013). Sensitivity and isolation are related to the challenge of validating every change in an independent manner. This is applied to our work as the policies adapt instances in an isolated manner without affecting the whole process. Thus, the verification is held in an isolation manner as well because the framework needs to verify the change on the current instance according to specific properties.

Incrementality expresses that validating a change does not mean to go back and check the validity of the previous change. In the context of the E-C-A policies, every time the policy changes the BPMN instance, the change is valid only for that current instance and does

not affect the other running instances. Therefore, the proposed framework verifies every change at the time of adaptation and verifying another change does not depend on other instances. Composability indicates the composition of two components is correct with the whole system. In this work we did not simulate the runtime environment where thousands of instances are running at the same time, so we did not try to validate two different instances at the same time.

It is also believed that the ability to automatically adapt a workflow instance while it is running imposes the challenge to provide an automatic and intelligent mechanisms that also able to validate correctness and consistency at runtime.

We showed how existing verification methods that are used in this thesis can manage goal compliance in self-adaptive workflows. Two different approaches were used for verification: (1) CSP trace refinement and (2) domain knowledge analysis through ontologies. However, the CSP trace refinement is stable and upfront process (defined at design time). It captures the property specification as derived from the goal specification. Thus, it is effective to ensure goal satisfaction when deleting a requirement at runtime unless the goal is changed. For verifying that new inserted or replaced requirements are compliant to the goal, different approach exploiting the ontologies is used. This is due to the fact that CSP is stable and cannot model the unknown requirements. Ontologies can compose the known and unknown requirements prior to runtime. Furthermore, they allow to semantically compose the domain knowledge by defining a semantic link among its vocabularies. However, ontologies cannot ever be complete and cover all the unknown requirements. Hence, we used the WordNet to overcome this shortcoming.

In terms of generalization, the proposed approach can be used for any workflow from any domain when requirements change is the case. Furthermore, the verification algorithms can be generalized to any application area (e.g. mission-critical systems) and our verification can run in parallel with other system-based verification (e.g. security check).

6.6 Summary

In this chapter, we evaluated the proposed framework and presented the results of the three evaluation aspects. The framework performance is measured in terms of the time taken by the framework to perform its functions. Two features are considered when measuring the time; BPMN complexity and reconfiguration complexity. We calculated the execution time by developing a Java code. The results showed that the execution time

increased when BPMN and reconfiguration complexity increased. The adequacy is evaluated based on the workflow patterns. Case by case analysis showed that the framework supports 25 out of 43 patterns. Furthermore, the proposed ontology is evaluated in terms of its accuracy including precision and recall. The results showed that the ontology is a promising technique for modelling BPMN domains and verifying them during their reconfiguration at runtime. Next chapter discusses this work in terms of capabilities and limitations and suggests further directions.

Chapter 7 Conclusion

7.1 Summary of Contributions

The goal-compliance constraints and the goal-compliance framework are developed in this thesis in order to provide assurances in self-adaptive workflows, BPMNs in particular. In the following, we discuss our contributions by chapters:

In Chapter 1, we presented the motivated context and related research challenges. The context presented some issues of validation and verification in the field of self-adaptive workflows which are affected by automated adaptation through E-C-A policies. Our primary aim was to provide assurances that any process adaptation through policies must be controlled under the original goal umbrella to exclude undesired behaviour. Hence, we presented our objectives to address the identified problem.

Chapter 2 presented the background technologies which we used to build up this work. It discussed the CSP transformation of BPMN diagrams and how to measure the satisfaction of the generic properties, such as deadlock and divergence freedom. It further analysed the related work to this research and highlighted the motivated research gaps.

In Chapter 3, we presented the goal-compliance framework, the motivated approach behind it and its implementation. The proposed framework has the ability to adapt BPMN structure as requested and validate the adaptation against the goal-compliance constraints. As the adaptation nature differs, we managed to study and analyse the effect of each adaptation logic in an independent manner while keeping in mind our original aim.

In Chapter 4, we analysed the effect of deleting tasks from BPMN instances on goal satisfaction. In this regard, we defined the goal-task dependency constraint which links goals from the goal model with tasks from the BPMN and allows a way of measuring goal satisfaction. In order to be able to establish a management link among goals and tasks, we defined a methodology to assign tasks to goals and translate that into properties to check their trace refinement with the adapted BPMN.

In Chapter 5, we analysed the effect of inserting new tasks into BPMN instances as well as replacing tasks with new tasks. The nature of inserting and replacing suggests to use ontology for verification purposes. Ontology is used to define extra vocabularies (tasks) as domain concepts and establish a semantic relationship among them. As a result, we

defined the domain-task conformance constraint and then further divided it to include the task-task consistency constraint. The former helps in identifying task inconsistency with a specific domain, while the latter helps in identifying task inconsistency with a specific task in a specific domain.

In Chapter 6, we consider evaluation the performance of the proposed framework and its applicability at runtime. The framework showed promising results as it took only a few seconds to complete its jobs – from reading specifications and reconfiguration to verification. The framework used two different verification methods: FDR trace refinement and ontology. Therefore, we evaluated the proposed ontology separately to figure out its accuracy. The results showed that the DGT ontology is accurate in most cases.

The outcomes of this study, explained in section 8.4, might contribute positively to any organisation that implements workflow systems, particularly BPMNs. Organisations will avoid catastrophic failures that are resulted from automated adaptation. Furthermore, the methodologies we defined will help business analysts to perform runtime verification of self-adaptive workflows. The methodologies (goal-compliance constraints) might be extended or reused to address the consistency issues in self-adaptive systems.

7.2 Discussions

This section discusses the research questions identified in section 1.7.

We asked whether goal satisfaction can be detected and checked with a high-level specification. This question is related to abstraction and uncertainty challenges. To what extent the abstraction of the goal and BPMN models to decide any undesired behaviour? Is the abstraction nature effective to handle and verify functional change? To answer the question, we will discuss the abstraction nature at both levels (i.e. goal and process). BPMN is considered a high-level specification capturing functional business requirements. It describes what to achieve and in what order while abstracted from other business and implementation details. For example, it does not contain information about the data flow. Functional requirements are originally elicited from goal specification and represented by the BPMN's activities. However, they are vulnerable to change at runtime as the business environment is highly dynamic and workflows are well-known for their complexity and dependency. E-C-A policies provide automatic adaptation at the business level in response to changing requirements. In similar way, the goal models capture the

stakeholders' objectives in terms of what abstracted from any other details on how to achieve them. The proposed goal-compliance constraints guarantee goal satisfaction at the business level as they help to identify undesired behaviour during reconfiguration at an early stage.

Most of the current approaches address the problem at a lower level, where interchanging data and dependency become more complex and may be hard to manage. What is more, the current focus in the field of self-adaptive systems verification is on assuring that the system is meeting its requirements while dropping/inserting new goals, as discussed in Chapter 2. We address the consistency issues during reconfiguration when the workflow changes its requirements and assume the goal is stable (unchanging). In Chapter 4 and 5, we show how the goal-compliance constraints help to identify any undesired behaviour and reject it.

We also asked how policies can be managed to exclude undesired behaviour while at the same time provide flexibility to cope with the kind of changes we wish to allow. To answer this question, we considered the argument about how to provide a flexible system while preserving its syntactic and semantic properties in order to preserve its quality. Our constraints successfully preserved the business outcome by disallowing any undesirable behaviour that violates the goal. However, the restrictions they imposed on the verification process might prevent valuable changes. On the other hand, increasing their flexibility could seriously affect business functionality. The task-task consistency constraint guarantees that the replace policy replaces only the equally semantic tasks. If we analyse their impact on verifying compliance, we can see that they prevented good changes at some points. For example, it was impossible to insert some tasks while they were compliant to the domain but the constraints were too restrictive. Hence, increasing the constraints flexibility without losing control is of utmost important to be considered. For example, rather than restricting the replace with the semantically equal tasks, it could be extended to consider relationships among business tasks in terms of their contribution to the same goal. In addition, the domain conformance constraint could be extended to handle complex task names instead of tasks of the Action-Object format. Furthermore, it could be enhanced with intelligent mechanisms in order to make a decision about whether this task is domain compliant or not. What is more, finding synonyms is sometimes not enough to decide task conformance. The framework could find synonyms for both parts

of the task (Action and Object) but the combination could violate the goal and cause unfavourable consequences. However, we suggest that the problem of restrictive constraints might be resolved through user intervention.

The constraints could also be enhanced with mechanisms that identify the tasks within each pool in the business process. For example, inserting 'Pay_Bill' into an organisation pool deviates from the goal because it represents a task that should be carried out by the customer, not the organisation.

We also asked how runtime adaptation can be guaranteed to meet the original requirements. In this regard, we found out that goal specification for BPMN models must be explicitly defined and linked to BPMN activities in order to be able to trace and measure them at runtime, as discussed in Chapter 4. Thus, we used the KAOS goal model which facilitated the establishment of a goal-task dependency link among the goal and BPMN models as well as DGT ontology, as discussed in Chapter 5. This helped to check the goal-satisfaction of the adapted BPMN at runtime. In connection with this point, we also asked whether understanding the domain from the goal model is enough to ensure goal satisfaction. Deriving BPMN domain concepts, which are representing BPMN tasks, and categorising them under predefined goals helped to manage the emerging functional requirements through policies. Although the proposed constraints do not take the temporal aspects into consideration, it is possible to handle them with both approaches. The proposed framework showed that the use of FDR, ontology and WordNet for verification purposes are promising for runtime verification.

The proposed framework was evaluated to show that it is effective for runtime verification. The development of the goal-compliance framework was to implement and test the goal-compliance approach. We did not use the Java profiling tool for evaluating the framework performance for two reasons:
(1) the framework is supposed to be embedded in a runtime environment but we could not do that as it is beyond the capabilities of this research, and
(2) our aim was to develop automatic and online verification capabilities for automatically adapted workflows and prove they are applicable in practice but not to develop a tool to outperform existing tools.

Chapter 6 discussed some of the main aspects to be considered when performing runtime verification and the goal-compliance framework showed its applicability to handle the runtime aspects with a promising performance.
Most of the test cases we ran through the goal-compliance framework showed that process goals are maintained against goal-compliance constraints. Some cases fail, as discussed in the previous chapter, and the proposed framework continues to run the original specification. It might be argued that this is inconvenient in practice and the framework should consider different alternatives. For example, the framework could be improved with intelligent techniques including learning and planning to enhance the decision-making process. We think that user intervention might be the immediate solution but not the best one as it could not be reliable with the automation and complexity nature of self-adaptive workflows.

We also asked questions regarding the existing verification strategies, weather they enable consistency assurance in self-adaptive systems. We think the adaptability nature of these systems restrict the ability of existing strategies to handle the challenge effectively. The FDR trace refinement captures the desired properties at design time when everything is planned and known prior to runtime. We could not use it to handle the insertion policies as their effect is not expected or unplanned. Therefore, we used the ontology to overcome this issue. However, the ontology also needs to be regularly updated to meet the requirements related to the adaptability nature.

7.3 Limitations and Further Research

This section sheds light on the limitations of our approach/framework and suggests future work.

7.3.1 Limitations

i. CSP and FDR

- FDR lacking AND notation as CSP language is a primitive language. For properties of type ‘AND-related tasks’, we could not define the property immediately as there is no CSP ‘AND’ operator, as discussed in section 4.5.1. To overcome this issue, we treat the CSP property specification for AND-related properties in the same way as the single task properties. Thus, we divided the

property specification into separate specifications. Each of them specifies the occurrence of each task individually. We can write a single property specification if the occurrence of all tasks is sequential. Otherwise, the property specification would become more complex.

- CSP cannot handle the insert and replace, DTC and TTC verification algorithms, as they need a fuzzy approach to manage them. CSP sematic helps for identifying syntactical and behavioural properties but does not have the functionality to deal with the adaptive-nature of workflows. Therefore, we used a different approach for managing the DTC and TTC verification algorithms as discussed in Chapter 5.

ii. Goal-compliance constraints

- The balance between the flexibility of the proposed constraints and the degree of consistency we wish to achieve is a challenging issue. In some cases, the constraints prevent desirable adaptation or validate undesirable adaptation, as discussed in the previous section. However, the proposed framework generates a feedback to the user in order to take the decision.

iii. Goal-compliance framework

- Policy conflicts issue to be handled when the policy is written.
- We developed a methodology to extract goal properties manually from the goal specification, as explained in section 4.4.1. However, it would be better to be automated. The goal's formal specification could be structured into an XML document or any other structured form to allow the framework to read it and extract the properties based on the methodology defined 'goal-task dependency link'.

7.3.2 Further Research

In the light of the above-mentioned limitations and the identified gaps in the related approaches, we suggest the following future work:

- A further study to improve the framework with intelligent techniques to enhance the verification process for deciding the consistency of inserting new requirements in a certain domain. Also, it is interesting to consider intelligent matching to deal with complex tasks' labels when using the ontology and WordNet

- A further study to map KAOS other models (Respect, 2007) to the BPMN for ensuring the consistency of other aspects.
- A further study to investigate whether our approach could be generalised for other type of systems, such as autonomous systems. A good start towards that might be by analysing the type of adaptation and study its impact on the behaviour of those systems with respect to the goal.
- A further study to improve policies to be able to learn from their behaviour to avoid repeating the undesired behaviour as policies lack the learning capabilities. Therefore, it would be interesting to refine or delete the policies which are trying to always make the same sort of change and always fail. This might help policies to be applied where they are really relevant.

7.4 Final Conclusion

The proposed framework along with the goal-compliance constraints address the goal-compliance issue in self-adaptive workflows. Automated adaptation can be managed in an autonomic manner to avoid unfavourable consequences in the face of complexity and automation. It is still a challenging to balance adaptation and the quality over adaptation. Due to the dynamicity and uncertainty nature of self-adaptive systems at runtime.

The key findings of this research can be summarised as follows:

- Moving the verification level to an abstract, early and user-friendly level leverages self-adaptive workflows to avoid inconsistency at runtime and avoid the complexity and dependency at lower data level.
- Self-adaptive workflows must be able to automatically reassess the adaptation as they are dynamic, dependent, knowledge-intensive and uncertain at runtime. In other words, reconfigurable systems must be able to interpret its current state, context and other information at runtime and may be enhanced with some intelligence to behave correctly according to these information. User intervention could be error prone, costly and time consuming

BIBLIOGRAPHY

UQU. Admission and Registration Guide for UQU. Available at: https://drive.uqu.edu.sa/_/dadregis/files/daleel2222.pdf (Accessed: 2014).

BPMN2Modeler. Available at: https://www.eclipse.org/bpmn2-modeler/ (Accessed: 2014).

JWI 2.4.0. Available at: https://projects.csail.mit.edu/jwi/ (Accessed: 2014).

Objectiver. Available at: http://www.objectiver.com/index.php?id=4 (Accessed: 2015).

OMG. Available at: http://www.bpmn.org (Accessed: 2014).

OWL API. Available at: http://owlapi.sourceforge.net (Accessed: 2015).

Protégé. Available at: https://protege.stanford.edu (Accessed: 2015).

Tools for CSP. Available at: https://www.cs.ox.ac.uk/publications/books/concurrency/tools/ (Accessed: 2014).

W3 Semantic Web. Available at: https://www.w3.org/OWL/ (Accessed: 2015).

WordNet. A Lexical Database for English. Available at: https://wordnet.princeton.edu/wordnet/frequently-asked-questions/for-application-developer/ (Accessed: 2015).

Abowd, G., Dey, A., Brown, P., Davies, N., Smith, M. & Steggles, P. (1999) 'Towards a better understanding of context and context-awareness', *Handheld and ubiquitous computing* Springer, pp. 304-307.

Abramowicz, W., Filipowska, A., Kaczmarek, M. and Kaczmarek, T. (2012) 'Semantically enhanced business process modeling notation', in Anonymous *Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications.* IGI Global, pp. 259-275.

Aguilar, J.C.P., Hasebe, K., Mazzara, M. and Kato, K. (2016) 'Model Checking of BPMN Models for Reconfigurable Workflows', *arXiv preprint arXiv:1607.00478.*

Ali, M. (2012) *Maintaining transactional integrity in long running workflow services: a policy-driven framework*. PhD thesis, University of Leicester.

Ali, R., Dalpiaz, F. and Giorgini, P. (2013) 'Reasoning with contextual requirements: Detecting inconsistency and conflicts', *Information and Software Technology,* 55(1), pp. 35-57.

Allehyani, B and Reiff-Marganiec, S (2017) 'Goal-Compliance Framework for Self-Adaptive Workflows', *The Ninth International Conference on Adaptive and Self-adaptive Systems and Applications* Copyright (c) IARIA, 2017, pp. 16-21.

Allehyani, B and Reiff-Marganiec, S (2015) 'Towards Ensuring a Correct Dynamic Adaptation of Workflows', *The Eighth Saudi Students Conference in the UK*World Scientific Publishing Company, pp. 123-132.

Allehyani, B. & Reiff-Marganiec, S. (2016) 'Maintaining Goals of Business Processes during Runtime Reconfigurations.', *ZEUS*, pp. 21-28.

Antón, A.I., McCracken, W.M. & Potts, C. (1994) 'Goal decomposition and scenario analysis in business process reengineering', *International Conference on Advanced Information Systems Engineering*Springer, pp. 94-104.

Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A. & Letier, E. (2010) 'Requirements reflection: requirements as runtime entities', *Software Engineering, 2010 ACM/IEEE 32nd International Conference on* IEEE, pp. 199-202.

Bizagi Suite. *BPMN by Example.* Available at: http://resources.bizagi.com/docs/BPMNByExampleENG.pdf (Accessed: 2014).

Blair, G., Bencomo, N. and France, R.B. (2009) 'Models@ run. time', *Computer,* 42(10).

Braubach, L., Pokahr, A., Jander, K., Lamersdorf, W. & Burmeister, B. (2010) 'Go4Flex: Goal-Oriented Process Modelling.', *IDC* Springer, pp. 7787.

Burmeister, B., Arnold, M., Copaciu, F. & Rimassa, G. (2008) 'BDI-agents for agile goal-oriented business processes', *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track* International Foundation for Autonomous Agents and Multiagent Systems, pp. 37-44.

Chatzikonstantinou, G. and Kontogiannis, K. (2016) 'Run-time requirements verification for reconfigurable systems', *Information and Software Technology,* 75, pp. 105-121.

Cheng, B.H., Eder, K.I., Gogolla, M., Grunske, L., Litoiu, M., Müller, H.A., Pelliccione, P., Perini, A., Qureshi, N.A. and Rumpe, B. (2014) 'Using models at runtime to address assurance for self-adaptive systems', in Anonymous *Models@ run. time.* Springer, pp. 101-136.

Cognini, R., Corradini, F., Gnesi, S., Polini, A. and Re, B. (2016) 'Business process flexibility-a systematic literature review with a software systems perspective', *Information Systems Frontiers*, pp. 1-29.

Computing, A. (2006) 'An architectural blueprint for autonomic computing', *IBM White Paper,* 31.

De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M. and Vogel, T. (2013) 'Software engineering for self-adaptive systems: A second research roadmap', in Anonymous *Software Engineering for Self-Adaptive Systems II.* Springer, pp. 1-32.

Fdhila, W., Rinderle-Ma, S., Knuplesch, D. & Reichert, M. (2015) 'Change and compliance in collaborative processes', *Services Computing (SCC), 2015 IEEE International Conference on* IEEE, pp. 162-169.

Feather, M.S., Fickas, S., Van Lamsweerde, A. & Ponsard, C. (1998) 'Reconciling system requirements and runtime behavior', *Proceedings of the 9th international workshop on Software specification and design* IEEE Computer Society, pp. 50.

Fredericks, E.M., DeVries, B. and Cheng, B.H. (2014) 'AutoRELAX: automatically RELAXing a goal model to address uncertainty', *Empirical Software Engineering,* 19(5), pp. 1466-1501.

Georgakopoulos, D., Hornick, M. and Sheth, A. (1995) 'An overview of workflow management: From process modeling to workflow automation infrastructure', *Distributed and parallel Databases,* 3(2), pp. 119-153.

Gorton, S.M. (2011) *Policy-driven Reconfiguration of Service-targeted Business Processes.* PhD thesis, University of Leicester.

Gorton, S., Montangero, C., Reiff-Marganiec, S. & Semini, L. (2007) 'StPowla: SOA, policies and workflows', *International Conference on Service-Oriented Computing* Springer, pp. 351-362.

Greenwood, D. (2008) 'Goal-oriented autonomic business process modeling and execution: Engineering change management demonstration', *International Conference on Business Process Management* Springer, pp. 390-393.

Guizzardi, R. & Reis, A.N. (2015) 'A method to align goals and business processes', *International Conference on Conceptual Modeling* Springer, pp. 79-93.

Hallerbach, A., Bauer, T. and Reichert, M. (2010) 'Capturing variability in business process models: the Provop approach', *Journal of Software: Evolution and Process,* 22(6-7), pp. 519-546.

Hallerbach, A., Bauer, T. & Reichert, M. (2009) 'Guaranteeing soundness of configurable process variants in Provop', *Commerce and Enterprise Computing, 2009. CEC'09. IEEE Conference on* IEEE, pp. 98-105.

Hlomani, H. and Stacey, D. (2014) 'Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey', *Semantic Web Journal*, pp. 1-5.

Horridge, M., Knublauch, H., Rector, A., Stevens, R. and Wroe, C., 2004. 'A practical guide to building OWL ontologies using the Protégé-OWL plugin and CO-ODE tools' edition 1.0. *University of Manchester*.

Jamwal, D. (2010) 'Analysis of software quality models for organizations', *International Journal of Latest Trends in Computing,* 1(2).

Jander, K., Braubach, L., Pokahr, A., Lamersdorf, W. and Wack, K. (2011) 'Goal-oriented processes with GPMN', *International Journal on Artificial Intelligence Tools,* 20(06), pp. 1021-1041.

Koliadis, G. & Ghose, A. (2006) 'Relating business process models to goal-oriented requirements models in KAOS', *PKAW*Springer, pp. 25-39.

Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G. and Becker, C. (2015) 'A survey on engineering approaches for self-adaptive systems', *Pervasive and Mobile Computing,* 17, pp. 184-206.

Kumar, A., Yao, W., Chu, C. & Li, Z. (2010) 'Ensuring compliance with semantic constraints in process adaptation with rule-based event processing', *International Workshop on Rules and Rule Markup Languages for the Semantic Web* Springer, pp. 50-65.

Leucker, M. and Schallhart, C. (2009) 'A brief account of runtime verification', *The Journal of Logic and Algebraic Programming,* 78(5), pp. 293-303.

Ly, L.T., Rinderle, S. and Dadam, P. (2008a) 'Integration and verification of semantic constraints in adaptive process management systems', *Data & Knowledge Engineering,* 64(1), pp. 3-23.

Mendling, J., Reijers, H.A. and van der Aalst, Wil MP (2010) 'Seven process modeling guidelines (7PMG)', *Information and Software Technology,* 52(2), pp. 127-136.

Müller, R., Greiner, U. and Rahm, E. (2004) 'Agentwork: a workflow system supporting rule-based workflow adaptation', *Data & Knowledge Engineering,* 51(2), pp. 223-256.

Nagel, B., Gerth, C., Post, J. & Engels, G. (2013) 'Kaos4SOA-Extending KAOS Models with Temporal and Logical Dependencies.', *CAiSE Forum*, pp. 9-16.

Natschlger, C. (2011) 'Towards a BPMN 2.0 ontology', *Business Process Model and Notation*, pp. 1-15.

Nurcan, S. (2008) 'A survey on the flexibility requirements related to business processes and modeling artifacts', *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual* IEEE, pp. 378-378.

Pang, S., Li, Y., He, H. and Lin, C. (2011) 'A model for dynamic business processes and process changes', *Chinese Journal of Electronics,* 20(4), pp. 632-636.

Parashar, M. and Hariri, S. (2005) 'Autonomic computing: An overview', *Unconventional Programming Paradigms*, pp. 257-269.

Pasquale, L., Baresi, L. & Nuseibeh, B. (2011) 'Towards adaptive systems through requirements@ runtime', *6th Workshop on Models@ run. time.*

Patig, S. and Stolz, M. (2013) 'A pattern-based approach for the verification of business process descriptions', *Information and Software Technology,* 55(1), pp. 58-87.

Peterson, J.L. (1981) 'Petri net theory and the modeling of systems'.

Pham, T.A. & Le Thanh, N. (2015) 'Ontology-based workflow validation', *Computing & Communication Technologies-Research, Innovation, and Vision for the Future (RIVF), 2015 IEEE RIVF International Conference on* IEEE, pp. 41-46.

Poels, G., Decreus, K., Roelens, B. and Snoeck, M. (2013) 'Investigating goal-oriented requirements engineering for business processes', *Journal of Database Management (JDM),* 24(2), pp. 35-71.

Radatz, J., Geraci, A. and Katki, F. (1990) 'IEEE standard glossary of software engineering terminology', *IEEE Std,* 610121990(121990), pp. 3.

Redlich, D., Blair, G., Rashid, A., Molka, T. and Gilani, W. (2014) 'Research challenges for business process models at run-time', in Anonymous *Models@ run. time.* Springer, pp. 208-236.

Regev, G., Soffer, P. and Schmidt, R. (2006) 'Taxonomy of Flexibility in Business Processes.', *BPMDS,* 236.

Regev, G. & Wegmann, A. (2005) 'A regulation-based view on business process and supporting system flexibility', *Proceedings of the CAiSE*, pp. 91-98.

Reichert, M. and Dadam, P. (1998) 'ADEPT flex—supporting dynamic changes of workflows without losing control', *Journal of Intelligent Information Systems,* 10(2), pp. 93-129.

Reichert, M., Rinderle, S. and Dadam, P. (2003) 'On the common support of workflow type and instance changes under correctness constraints', *Lecture notes in computer science*, pp. 407-425.

Reichert, M. and Weber, B. (2012) *Enabling flexibility in process-aware information systems: challenges, methods, technologies.* Springer Science & Business Media.

Respect, I.T. (2007) 'A KAOS tutorial'. Available at: http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf (Accessed: 2015)

Rinderle, S., Reichert, M. and Dadam, P. (2004) 'Correctness criteria for dynamic changes in workflow systems—a survey', *Data & Knowledge Engineering,* 50(1), pp. 9-34.

Rinderle-Ma, S. (2009) 'Data flow correctness in adaptive workflow systems', *Emisa Forum*, pp. 25-35.

Roscoe, B. (1998) 'The theory and practice of concurrency'.

Russell, N., Ter Hofstede, A.H., Van Der Aalst, Wil MP and Mulyar, N. (2006) 'Workflow control-flow patterns: A revised view', *BPM Center Report BPM-06-22, BPMcenter.org*, pp. 6-22.

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

Saidani, O. & Nurcan, S. (2007) 'Towards context aware business process modelling', *8th Workshop on Business Process Modeling, Development, and Support (BPMDS'07), CAiSE*, pp. 1.

Salehie, M. and Tahvildari, L. (2009) 'Self-adaptive software: Landscape and research challenges', *ACM transactions on autonomous and adaptive systems (TAAS),* 4(2), pp. 14.

Santos, E., Castro, J., Sanchez, J. & Pastor, O. (2010) 'A Goal-Oriented Approach for Variability in BPMN.', *WER*, pp. 17-28.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N. & van der Aalst, Wil MP (2008) 'Towards a Taxonomy of Process Flexibility.', *CAiSE forum*, pp. 81-84.

Sell, C., Winkler, M., Springer, T. and Schill, A. (2009) 'Two dependency modeling approaches for business process adaptation', *Knowledge Science, Engineering and Management*, pp. 418-429.

Szvetits, M. and Zdun, U. (2016) 'Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime', *Software & Systems Modeling,* 15(1), pp. 31-69.

Tamura, G., Villegas, N., Müller, H., Sousa, J.P., Becker, B., Pezze, M., Karsai, G., Mankovskii, S., Schafer, W. and Tahvildari, L. (2013) 'Towards practical runtime verification and validation of self-adaptive software systems'. *In Software Engineering for Self-Adaptive Systems II* (pp. 108-132). Springer, Berlin, Heidelberg.

Trcka, N., Van der Aalst, Wil MP & Sidorova, N. (2009) 'Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows.', *CAiSE*Springer, pp. 425-439.

Van Der Aalst, Wil MP and Basten, T. (2002) 'Inheritance of workflows: an approach to tackling problems related to change', *Theoretical Computer Science,* 270(1-2), pp. 125-203.

van der Aalst, Wil MP, Dumas, M., Gottschalk, F., Ter Hofstede, A.H., La Rosa, M. and Mendling, J. (2010) 'Preserving correctness during business process model configuration', *Formal Aspects of Computing,* 22(3-4), pp. 459-482.

van der Aalst, Wil MP and Jablonski, S. (2000) 'Dealing with workflow change: identification of issues and solutions', *Computer systems science and engineering,* 15(5), pp. 267-276.

van Der Aalst, Wil MP, Pesic, M. and Schonenberg, H. (2009) 'Declarative workflows: Balancing between flexibility and support', *Computer Science-Research and Development,* 23(2), pp. 99-113.

van Lamsweerde, A. (2004) 'Goal-oriented requirements engineering: a roundtrip from research to practice [engineering read engineering]', *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International* IEEE, pp. 4-7.

Van Lamsweerde, A. (1991) 'The KAOS Project: Knowledge Acquistion in Automated Specification of Software', *Proc. AAAI Spring Symposium Series, Stanford University, American Association for Artificial Intelligence, March 1991*.

Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L. & Casallas, R. (2011) 'A framework for evaluating quality-driven self-adaptive software systems',

Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems ACM, pp. 80-89.

Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M. and Ghner, P. (2014) 'Challenges for software engineering in automation', *Journal of Software Engineering and Applications,* 7(05), pp. 440.

Weber, M. and Kindler, E., 2003. The petri net markup language. In *Petri Net Technology for communication-based systems* (pp. 124-144). Springer Berlin Heidelberg.

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

Wieland, M., Kopp, O., Nicklas, D. & Leymann, F. (2007) 'Towards context-aware workflows', *CAiSE07 Proc. of the Workshops and Doctoral Consortium*, pp. 78.

Wong, P. (2011) *Formalisations and Applications of Business Process Modelling Notation.* PhD thesis*,* University of Oxford.

Wong, P.Y. & Gibbons, J. (2009) 'Property Specifications for Workflow Modelling.', *IFM* Springer, pp. 56-71.

YongLin, X. & Jun, W. (2008) 'Context-driven Business Process Adaptation for ad hoc changes', *e-Business Engineering, 2008. ICEBE'08. IEEE International Conference on* IEEE, pp. 53-60.

Zander, Thorsten, Krol & Laurens (2016) *The Potential of Automated Adaptation.* Available at: http://neuroadaptive.org/blog/the-potential-of-automated-adaptation.