2016

A Framework for Semantic Effect Annotation of Business Process Models

Kerry G. Hinge
University of Wollongong

A Framework for Semantic Effect Annotation of Business Process Models

Kerry G. Hinge

Supervisor:
Professor A. Ghose
Co-supervisor:
Dr. H. Khan Dam

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

The University of Wollongong
School of Computing and Information Technology

August 2016

Declaration

I, Kerry G. Hinge, declare that this thesis submitted in partial fulfilment of the requirements for the conferral of the degree Doctor of Philosophy, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Kerry G. Hinge
October 8, 2017

Abstract

A key challenge in devising solutions to a range of problems associated with business process management, such as process life cycle management, compliance management and enterprise process architectures, is the problem of identifying process semantics. The current industry standard business process modelling notation, BPMN, provides little by way of semantic description of the effects of a process (beyond what can be conveyed via the nomenclature of tasks and the decision conditions associated with gateways). This thesis describes the conceptual underpinnings, design and implementation of the Process Semantic Effect Evaluation and Reasoning (ProcessSEER) framework. The framework supports several strategies for obtaining semantic effect descriptions of BPMN process models without imposing the overly onerous burden, of providing formal specification, on the business analyst. As a validation of the framework, a tool has been implemented that requires business analysts to describe the immediate effects of each task in a process model. These are then accumulated in an automated fashion to obtain cumulative effect annotations for each task in a process. The tool leverages, and contributes to, domain ontologies wherever they are available. The tool also permits the business analyst to specify immediate effect annotations in a practitioner-accessible format that enables formal specification using a limited repertoire of natural language formats. An example is provided of the tool being used in a logistical setting showing the benefits of semantic effect annotation. Further research findings reveal how crowdsourcing techniques can be utilised to broaden the scope of the tool's knowledge acquisition capabilities. Applied to a clinical environment the tool is shown to detect treatment conflicts between multiple treatment protocols that are co-incident (on the same patient). Detection is a difficult and open problem that is particularly exacerbated in the context of treating multiple medical conditions co-occurring in aged patients. For example, a clinical protocol for prostate cancer treatment requires the administration of androgen-suppressing medication. This could negatively interact with another, co-incident, protocol, if the same patient were being treated for renal disease via haemodialysis, where androgen-enhancers are frequently administered. Treatment conflicts such as these are subtle, and usually difficult to detect using automated means.

Dedication

I dedicate this thesis in loving memory of my father and mother,
George and Gwennyth Hinge,
without whose assistance this would not have been possible.

Acknowledgments

My sincere gratitude goes to Professor Aditya Ghose for his continued support and understanding through my difficult circumstances.

Contents

List of Publications

E. D. Morrison, A. K. Ghose, H. K. Dam, K. G. Hinge, and K. Hoesch-Klohe, "Strategic alignment of business processes," in WESOA'11. Proceedings of the 7th International Workshop on Engineering Service-Oriented Applications, December 2011.

K. Hinge, A. Ghose, and A. Miller, "A framework for detecting interactions between co-incident clinical processes," International Journal of E-Health and Medical Communications, vol. 1, no. 2, pp. 2435, 2010.

K. Hinge, A. Ghose, and G. Koliadis, "Process SEER: A tool for semantic effect annotation of business process models," in Proceedings of the Thirteenth IEEE International EDOC Conference. IEEE, September 2009

List of Figures

List of Algorithms

Chapter 1

Introduction

Modern day visionaries like Tim Berners-Lee have painted us a picture of the Semantic Web [17], an environment in which intelligent agents can negotiate on our behalf and in general make interaction with the internet more meaningful. This prediction initially caused a great deal of excitement and much effort was invested towards achieving this design but the reality of its construction soon became apparent. Many were unwilling to invest the necessary time into something that had questionable returns on investment. From an economic perspective the semantic web seems like a lot of hard work for little reward. In fact many of the benefits attributed to the Semantic Web apply to consumers rather than the people investing in its development. This is one possible reason why the business community have not enthusiastically engaged in the semantic mark-up of their existing web sites.

Apart from the extra work required to add semantic mark-up to web sites, the terminology used must be referenced to an ontology. An ontology is a relationship model describing relationships between words as they are used within a specific domain [135]. These context specific descriptions and relationships act as an explicit definition of the meaning of a word. Manually building an ontology requires a lot of time and effort and further adds to the cost of Semantic Web development [69]. Given that the main purpose for semantic mark-up is the retrieval of more accurate information then the business case scenario does not appear to be very profitable. If the Semantic Web is ever to become a reality then it will certainly require world-wide adoption by the business community. For that to happen, there needs to be a greater incentive for business to embrace the development of this infrastructure.

An area of research that has attracted a great deal of business interest and investment is Enterprise Resource Planning (ERP). The implementation of an ERP system requires all business processes within an organisation to be modelled [56]. A process model is a visual representation of the order in which actions are performed (see Fig:1.1). The visualisation evokes recognition more effectively (with regard to communicating information) than text alone. This improved recognition translates

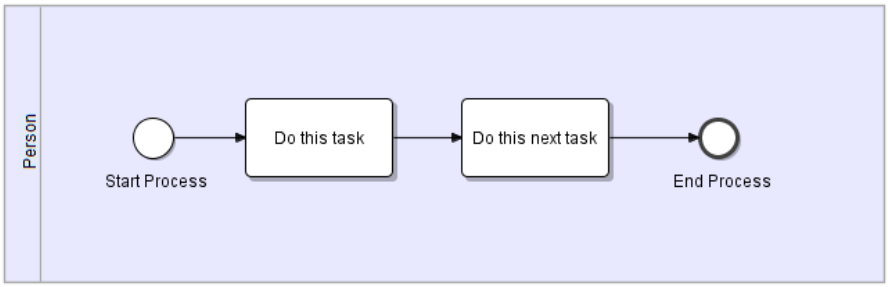

Figure 1.1: A simple business process model depicting a sequence of two activities.

into more efficient methods of communicating instructions. The purpose of the model is to maintain consistency in process execution. The visualisation of a process also assists with process analysis. Components of the model can be easily moved around to experiment with different configurations to achieve optimal efficiency.

In most cases process models provide a managerial perspective of workflow within an organisation. They communicate the flow of activities and the movement patterns of employees in the workplace. Process models can be optimised for efficiency and quickly checked for compliance. From a business analyst perspective this is by far the most common use of a business process model. A manager, on the other hand, will consult a business process model and issue instructions to employees. An IT specialist may translate a business process model into execution code such as WSBPEL [14] to coordinate web services. An employee could conceivably use a business process model as an instruction manual to explain the how-to of a particular job. These different uses produce subtly different process models that are explored further in section 4.8.

Business process models are limited by the quality of information they can communicate which often leads to costly errors in judgement occurring as a consequence of optimisation initiatives. In [80], Koliadis et al. describe a method of adding semantic effect annotations to business process models with the intent of improving process change within an organisation. There are definite cost-saving benefits, with regard to optimisation efforts, that are associated with this form of semantic mark-up. If these semantically annotated business process models could be used to generate Semantic Web content then an economic incentive would exist for business participation in this development. Koliadis et al. are describing a reasoning engine for processing the annotations if they existed. Ghose et al. expand upon this idea [52] by defining an explicit grouping of tasks within a business process model called a *Scenario Label*. This grouping acts as a structure for accumulating semantic annotations so they can be processed by the reasoning engine. The intent behind this is to automate the analysis of business process models to assist analysts with design-

time decisions. Although the structure is defined, the method of accumulation is not.

The opportunity therefore exists to build an application that can annotate business process models and accumulate the semantic annotations into a format that can be machine interpreted. Given that the business community will be producing semantically annotated models with such an application the next logical step will be to extend the functionality to convert the business process models into Semantic Web content [86, 151, 130, 25]. Such an application will present an attractive opportunity for the business community to become actively involved with the Semantic Web and consequently hasten its development.

The application proposed by Ghose and Koliadis focuses on supplying real-time analytics to improve strategies like business process design, change management consistency checking and compliance management. It is not sufficient to simply annotate business process models with semantics. A reasoning engine must have machine interpretable annotations before it can process and analyse the data. Writing semantic effect annotations in formal logic is not an attractive proposition for business analysts who have no training in that discipline. This research investigates alternate solutions to overcome this problem and identifies a critical component necessary for the success of the proposed application.

If the effect annotations are going to be truly semantic then the words used in the annotations must be linked to an ontology so that their meaning can be verified. This implies that the proposed application must include ontology management capabilities. We investigate some ontology tools and current industry standards under development to arrive at a solution for what has now become a proposed application suite.

The final key ingredient required for the suite of applications is the ability to convert from semantically annotated business process models to Semantic Web content. The particular area of interest is in the domain of Web Services. Web Services are independently operating web applications that can collaborate to perform complex business processes. Software already exists that can translate BPM's into Web Services but not Semantic Web Services. The latter requires semantically annotated business process models to achieve this goal. The Semantic Web content that is most likely to attract business interest will be Semantic Web Services because of its ability to automate existing business processes and interface with customers and suppliers online. The set of tools that can achieve this will not only encourage the development of the Semantic Web but also improve business process management. There is a trend towards an integrated development environment [109]. The tool suite described in this document is a positive step towards this type of integrated development.

Annotating and analysing specifications of program functionality, in order to help establish program correctness, has a long tradition dating back to the introduction of the axiomatic techniques proposed by Hoare and Dijkstra [74]. With sufficient information, these forms of annotations [137] provide a basis for answering questions relating the identification of: the conditions enabling a process to be performed (i.e. postdiction); the conditions resulting from a process being performed in some context (i.e. prediction); and, the processes with the capability of realising a set of conditions when executed in some context (i.e. planning). Recently, similar proposals have emerged in the domain of web services [104] [95]. These forms of specification can be effective for performing analyst related tasks, however their utility and availability in some situations can be limited (e.g. cost restrictions) – warranting a need for "partiality" and "lightweight" approaches [76]. The contribution in this document are techniques to leverage a partial specification of functional effects annotated to business process models. In exploring this space we asked the following research questions:

RQ1 Is it possible to provide richer semantics for business process models via effect annotations?

RQ2 Is it possible to determine, at design time, for any point in a process design, the effects that a process would achieve if it were to execute up to that point?

RQ3 Is it possible to build a robust tool to support business analysts in developing such annotated process models?

RQ4 Is it possible to support analysis tasks such as goal satisfaction, compliance checking, semantic conformance checking and semantic simulation?

RQ5 Is it possible to crowd source these annotations?

RQ6 Do compelling use cases for this technology exist in use cases such as medicine?

The work presented in this document follows a Design Science model described by Hevner et al.[62] consisting of three research cycles, the *relevance cycle*, the *rigor cycle* and the *design cycle*. Answering the questions above establishes the *relevance* of this research and its potential contribution to a real world environment. The term 'Design Science' evolved from the work of Herbert A. Simon in his treatise "Sciences of the Artificial" [140] that explored the underlying "science of design" and stressed the importance of real world application in any scientific research. In revealing the science involved in design Simon crosses the boundaries between science and engineering, showing that the same techniques used to investigate the natural world are also applicable to the exploration of the artificial.

The Background chapter of this document provides the reader with information about the different technologies underpinning this research. Many of the research questions could be answered with a disparate selection of technologies but the focus here has been on a solution that supports the interoperability of such technologies with a distinct focus on complying with global standards. This chapter presents the results of *rigorous* research into the complex web of often competing technologies that have contributed to a solution to the problems and opportunities identified in chapter 1. Also discussed are issues like why we need semantic annotations and which technologies are necessary to establish a practitioner-accessible framework. A review of related work in this field with comparisons to this research conclude the Background chapter.

In chapter 3 the basic building blocks of the framework for semantic effect annotation of business process models are introduced. Process effect accumulation has another unique set of problems that are also identified in this chapter. Solutions to these problems are discussed and supported by a set of algorithms that can be found in Appendix A. The algorithms support a variety of accumulation methods over a select group of process model instances. The concept of effect scenario labels is also introduced as a control mechanism for algorithm selection. All algorithms contribute to a fundamental pair-wise effect accumulation procedure that can adjust to different process model configurations. These effect accumulation algorithms co-participate with a background reasoning engine to derive alternate outcomes for any point in a process model. Chapter 3 concludes with an extensive example of the framework being used in a warehouse delivery service.

The *design cycle* mentioned in [62] is captured in chapters 3 and 6 where the technologies discussed in chapter 2 are combined into a cohesive design developed over several iterations in which the application software was tested in a real world environment. Chapter 4 reports on the outcome of that testing in a medical environment. One element of the final application suite is effectively implemented and used to model medical procedures. The chapter looks at the practicality of the framework in real-life situations and in some cases life and death situations. Also included in the chapter is a case study from which valuable insights are gained that contribute to critical changes in application development, and to the underlying framework. The potential application of the framework in dynamic workflow management systems that support mobile instructional devices is also explored. It is worth noting that although a clinical environment was chosen for testing, the framework has broad reaching applications across a variety of domains.

Chapter 5 explores the potential of utilising crowdsourcing techniques as a means of expanding or refining a knowledge base to improve its content. Techniques are described showing how this framework can contribute to the enrichment of a

domain ontology. In chapter 6 the main component in the application suite has been implemented. The implementation of the conceptual elements from chapter 3 is described along with the user interface design. The chapter also describes how the application can be linked to an ontology and how it can directly output an ontology for knowledge acquisition purposes.

The thesis concludes with a summary of the work so far with recommendations for future research and development. Appendix A contains a complete list of all the algorithms described in chapter 3. Appendix B lists standards and specifications controlling application development. URL's are also provided for accessing associated documents. Appendix C lists some of the governing organisations involved with Semantic Web related activities. Appendix D lists applications referred to throughout this document with associated URL's.

Chapter 2

Background

Hypertext Markup Language (HTML) and Extensible Markup Language (XML)[116] emerged from an international standard (ISO8879) called Standard Generalised Markup Language (SGML). HTML was adopted for the presentation of information on the internet. Its purpose is to specify different elements of a web page and define how they are displayed in a web browser. HTML uses *tags* that encapsulate data. For example, `<html>` is an opening tag and `</html>` is a closing tag. Together they specify that the content between the two tags is a HTML document. The first mention of HTML on the internet was in a correspondence by Tim Berners-Lee in 1991 [19] in which he references the first internet published document on HTML Tags [18]. HTML has a defined set of tags with which a compliant web browser can correctly display an HTML document.

XML is syntactically similar to HTML but is not constrained by a fixed set of tags. Users of XML are free to name their own tags but they must also provide an interpreter to make sense of any newly created tags like a web browser interprets HTML. XML provides metadata about the information it contains. For example, `<author>John Smith</author>` tells us that John Smith is an author. It could be said that HTML describes what to do with the information whereas XML describes how to interpret the information. Both HTML and XML are global standards maintained by the World Wide Web Consortium (W3C). They are platform independent, providing a structured framework for the global exchange of information.

A key consideration when developing a framework for semantic effect annotation of business process models is for businesses to be able to communicate freely without having to purchase expensive middleware to translate between organisations. It is therefore important that selected technologies are compliant with these international standards so that information can be easily exchanged over the internet. The principal technology employed in this work is Business Process Model and Notation (BPMN) which provides a mapping to XML.

2.1 Business Process Model and Notation

The idea of recording and documenting work practices can be traced back to the work of Frederick W. Taylor and Henry L. Gantt in the early 1900's. Taylor introduced the concept of scientific management in 1911 in his work *"The Principles of Scientific Management"*[145]. It is attributed to being the first scientific study of work practices. The motivation behind this was to discover what is now commonly referred to as *"Best Practice"*. Gantt's contribution [47] to the field of project management is still in use today, the *"Gantt Chart"*[155], a bar graph allocating jobs and time completion deadlines.

Post WWII, Japan's focus on *Total Quality Control*, fuelled by the work of Kaoru Ishikawa[75], gave it a significant competitive advantage in the global market. The notion of *best practice* had evolved into a system of management. A number of *Quality Management Systems* (QMS) have gained acceptance over the years like *Total Quality Management* (TQM) and Six Sigma. These systems have driven the demand for process documentation.

In 1987 the ISO9000 series of standards for quality assurance was published. Many organisations could not win contracts unless they were ISO9000 certified. An important part of the ISO9000 certification involves the documentation of all business processes. Documentation was regarded as a measure of repeatability such that quality standards could be maintained but many organisations, having attained certification, simply filed the process documentation away. Documented processes in a textual format take considerable time to read before the underlying idea of the process can be understood. On the other hand, a process model, using graphical symbols to represent different elements of a process, proved to be a far more intuitive means of communicating process ideas.

Process modelling is just one type of documentation in the quality management toolkit. It dates back to as early as 1921 when the *Flow Process Chart* was introduced to the members of the American Society of Mechanical Engineers (ASME) by Frank and Lillian Gilbreth[53]. A *Flow Process Chart* is a graphical representation of process activities based on a rudimentary set of symbols. The *Flow Process Chart* spawned many different variations of *Flowcharts* including, but not limited to, Workflow Charts, Data Flowcharts, Program Flowcharts, System Flowcharts, Document Flowcharts and Process Flowcharts.

By the 1990's many different modelling notations were in existence and in the field of software engineering this caused widespread incompatibility issues. In response the Object Management Group (OMG) facilitated discussions between industry professionals to develop a standard visual software modelling notation. The Unified Modelling Language (UML) [114] was adopted and published by the OMG

in the mid 1990's. Standardisation meant that code could be generated from UML based software modelling tools, further improving productivity.

Business Process Management (BPM) also suffered from a fragmented market of modelling notations. A group of professional organisations, many of whom had contributed to the fragmentation of the business process modelling market, came together to form the Business Process Management Institute (BPMI) whose agenda was to standardise a business process modelling notation that was based on an executable language. The outcome of this collaboration was the release of the Business Process Model and Notation (BPMN)[112] version 1.0 in May 2004. The BPMI was subsumed by the OMG in February 2006 who in turn have continued the development and maintenance of the BPMN standard.

The development of the BPMN standard was motivated by a need to develop a modelling notation that could bridge the gap between process design and process implementation and translate easily into executable code, notably Business Process Execution Language for Web Services (BPEL4WS) [154]. While other modelling notations have proven effective at modelling programming code artifacts, in the case of UML, or providing an intuitive visual representation for describing a process, in the case of Workflow Diagrams, BPMN combined both these attributes into a single notation. A BPMN model can therefore be used as a visual instructional tool for human consumption and as a template for automated code generation. When process segments are identified for automation then a programmer can use the BPMN model to automatically generate code. This is an important economic consideration when planning for future automation of processes that are currently performed manually. BPMN is an evolving standard (currently at version 2.0.2) that is continually growing in expressiveness through consultation with domain experts.

For further information about process modelling notations Rosemann et.al.[121, 122, 57] have evaluated various notations such as Petri nets, Event Driven Process Chains and BPMN using quality benchmarks. In [127] Rosemann et.al. conduct a survey of organisations about their business process management (BPM) service requirements and capabilities, identifying a comprehensive list of 15 potential services for effective business process management. The framework presented in this document provides a foundation for many of the services identified in [127].

BPMN defines a large number of icons that can be used as nodes and a smaller number of edge styles. It is now a popular method for defining business processes that can be used as a visual representation of what either people or software applications do when performing a business process. The notation uses a broad selection of graphical symbols to represent business process elements. Only a limited subset of BPMN symbols have been implemented for this project. Future implementations are intended to utilise the complete set of BPMN symbols. The subset of symbols

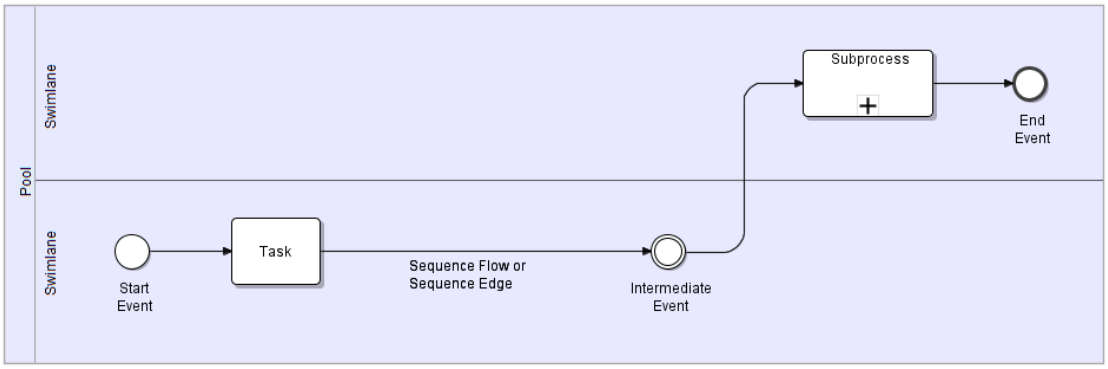

Figure 2.1: BPMN Elements: Pools, Swimlanes, Tasks, Start Events, End Events, Intermediate Events and Sequence Flows/Sequence Edges.

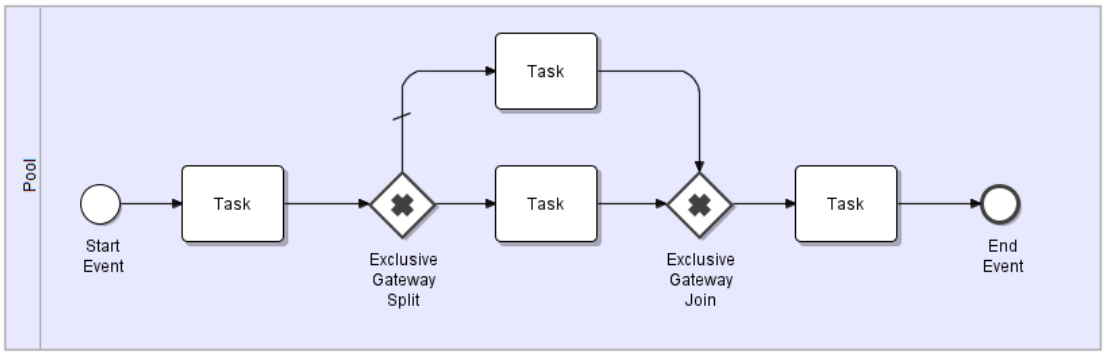

Figure 2.2: BPMN Elements: Exclusive Gateways.

that were used in this implementation include Pools, Empty Start Events, Empty End Events, Empty Intermediate Events, Tasks, Sequence Flows/Sequence Edges, Parallel Gateways, Exclusive(XOR) Gateways and Inclusive Gateways.

A Pool (Fig:2.1) represents an actor that is performing a task. The actor may be a person, a department, an organisation or even a software application. The pool metaphor arises from the shape representing an Olympic swimming pool. A pool is a boundary device encompassing all activities performed by a single actor. An actor can also be a group so Swimlanes are used to separate the individuals in the group.

Tasks, (Fig:2.1) define the actions performed by an actor. Each Task can either describe what the actor is doing (declarative) or be an instruction indicating what the actor must do (imperative) at a particular stage in the overall process. Tasks are connected by Sequence Flows, also referred to as Sequence Edges. These are solid lines with arrows that indicate the direction and order in which Tasks are performed or in which Events occur.

Gateways (Figs:2.2, 2.3, 2.4) are used to either split or join Sequence Flows. In the case of an exclusive gateway (Fig:2.2) it could represent a decision where if a certain condition is met then sequence flow would continue one way but if the condition was not met then the sequence flow would continue the other way. An

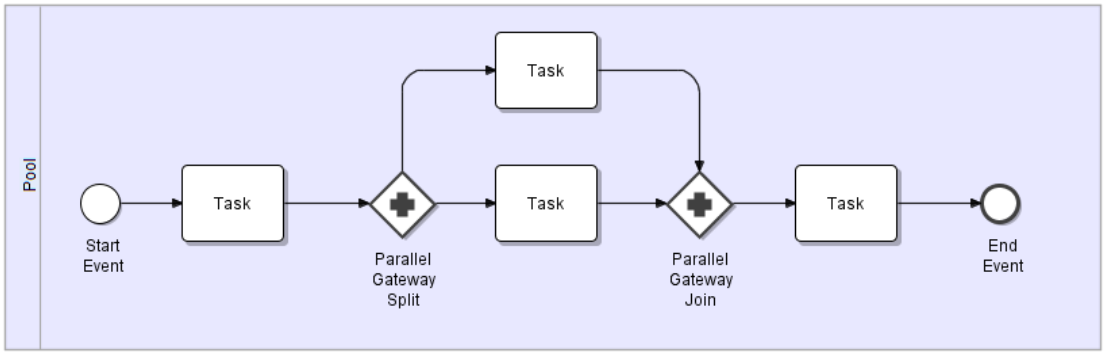

Figure 2.3: BPMN Elements: Parallel Gateways.

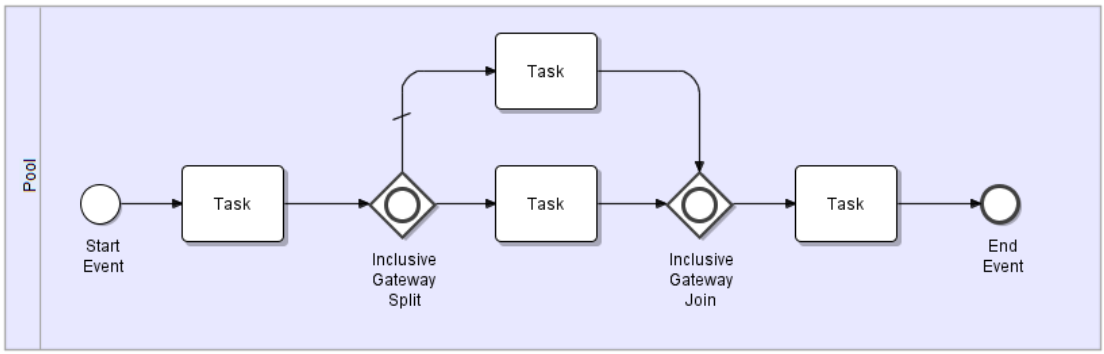

Figure 2.4: BPMN Elements: Inclusive Gateways.

inclusive gateway (Fig:2.4) is also representative of a decision. The difference being that sequence flow can continue along more than one branch depending on how many conditions are satisfied. Parallel gateways occur when the actor represents a group of people or an application running concurrent threads. The implication is that the activities occurring on each parallel Sequence Flow are occurring at the same time but this may not be the case and is an assumption that is not consistent with BPMN. In this document the term *Gateway Structure* is used to refer to everything that occurs between and including a gateway split and a gateway join. Typically a *Parallel Gateway Structure* will include both the parallel gateway split and join and everything between them including any nested gateway structures.

Finally, Start and End Events (Fig:2.1) simply denote the beginning and the end of a process while Intermediate Events occur within the process model structure. They can indicate certain conditions that exist but in the context of this document only Empty Events are used. Events indicate actions that occur outside the control of an actor. They can trigger actions or be the consequence of actions. In this document Tasks and Events are generically and collectively referred to as either activities or actions. Further information regarding BPMN can be found in the "BPMN Modeling and Reference Guide" [153].

A business process model is a flowchart and in its most abstract representation, a directed graph of nodes and edges. A formal definition of *Scenario Labels*, a graph based encoding of business process models, is provided in [63, 64]. A *Scenario Label* is a sequence of activities that maps a single executable path through a process model. When a task in a process model is selected, all possible paths that lead to the selected task are mapped to *Scenario Labels*.

The growing interest in business process management (BPM) [2] methodologies and tools, as well as the high levels of adoption of such technologies in industry has led to a greater need for more sophisticated techniques for analysing and reasoning with business process models. Much of the analysis required for process compliance management [51], change management [80], enterprise process architectures [61], strategic alignment [106], process improvement [65] and the management of life cycles and inter-operation of business processes [83, 88] relies on being able to refer to the *semantics* of those processes. BPMN as well as several other similar notations, provide a means for describing the *coordination semantics* of business processes but not the semantics of processes in terms of their *effects*. Thus a BPMN process model might require that task A must precede task B, but does not provide any indication of what is done by tasks A and B (beyond what might be implicit in their nomenclature), i.e. their effects. It is difficult to determine from a process design in BPMN what the effects achieved by a process might be at any point in the process design.

This is one of the problems that this document seeks to address. The problem is not alleviated by taking recourse to the formal semantics of process design notations such as BPMN. Such semantics, as pointed out above, only describe the coordination aspects of a process. In addition, there is no consensus on the semantics of BPMN [3]. The "ProcessSEER" tool, presented in [63], provided the capability to annotate effects to BPMN *tasks* and *events* (henceforth also generically referred to as *actions or activities*). The tool is also capable of accumulating effects across a process model and reasoning about those effects thus allowing it to answer the question, "What would be the result if the process had executed up to this point?". Version 1.0 of ProcessSEER accumulated effect annotations by stepping through Scenario Labels and performing pair-wise accumulation at each step. The current version of Process SEER dispenses with Scenario Labels and accumulates at each activity as it walks through the process model.

In 1998 Wil van der Aalst[1] proposed the use of Petri nets for modelling workflow, noting their formal semantics. BPMN is based upon Petri nets with much of the BPMN documentation referring directly to Petri nets to explain execution semantics. Petri nets have four graphical elements, a *Place*, a *Transition*, an *Arc* and a *Token*. A *Token* can represent a product or a state. Process execution is

represented by *Tokens* being passed along *Arcs* between *Places* and *Transitions*. The properties of Petri nets underpin how semantic effects are accumulated across a business process model. Petri nets have also influenced how and where semantic effects are annotated which is explained later in the document.

2.2 Petri Nets

Petri Nets [124] consist of four elements:

- Transitions
- Places
- Tokens
- Arcs

Transitions
A *Transition* is equivalent to an action or activity or event or even a gateway on a BPMN process. They consume *Tokens* and generate *Tokens*. *Tokens* do not pass through them. The consumption of a *Token* is equivalent to the satisfaction of a precondition.
Places
Places are place holders for states that exist prior to or following a *Transition*. They hold *Tokens*.
Tokens
Tokens are representative of conditions or states. When a *Token* is in a *Place* it means that a precondition exists for the following *Transition* to execute.
Arcs
Arcs connect *Places* to *Transitions* and *Transitions* to *Places*. They are directional, indicating the direction in which *Tokens* are passed between *Places* and *Transitions* and visa-versa. *Arcs* cannot connect a *Transition* to another *Transition* or a *Place* to another *Place*. *Arcs* also represent the number of *Tokens* that can be passed along them. The number of *Tokens* passed along each *Arc* is dependent upon the weight of the *Arcs*. If an *Arc* has a weight of 2 then 2 *Tokens* WILL be passed along that *Arc*. If 2 *Tokens* do not exist in the preceding *Place* then no *Tokens* will be passed. The weighting of *Arcs* exiting a *Transition* indicates how many *Tokens* the *Transition* will generate and pass along that *Arc* when it fires.

Petri Nets and BPMN

A Petri net represents a business process model in the following way. An activity requires certain preconditions before it can be executed. The *Place* preceding that

Activity/*Transition* holds *Tokens* and each *Token* represents a single precondition. When the required number of *Tokens*/Preconditions exists in the preceding *Place* then the Activity/*Transition* is Fired/Executed. When the Activity/*Transition* is Fired/Executed it consumes all the *Tokens*/Preconditions and generates a new collection of *Tokens*/Effects that are transferred to the *Place* following the Activity/*Transition*. In this case there is no difference between a precondition and an effect. The *Token* is a common symbol used for both. What was an effect generated by a previous *Transition* becomes a precondition for the following *Transition*. In the case of a Gateway/*Transition*, they have two or more *Arcs* either exiting or entering them.

2.3 Workflow Management Systems

The Workflow Management Coalition (WfMC) is a global consortium of consultants, vendors and academics who promote and develop standards for the terminology and interoperability of workflow applications. The Workflow Reference Model [67] was published by the WfMC in 1995 and is still the model on which most workflow software systems are built today. In [67], Hollingsworth defines workflow as:

> "The computerised facilitation or automation of a business process, in whole or part".

Business process models describe workflow by the coordination made explicit in their structure. However, they only specify the order of task execution, not the timing. A Workflow Management System (WfMS) is defined in [67] as:

> "A system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. Such a system maps to the Workflow Coalition's reference model" [67].

A WfMS does not actually perform any of the tasks in a business process. It coordinates and supports the timely execution of tasks and the provision of resources [123]. In the context of this research, semantic effects are utilised by a WfMS to monitor and evaluate the current conditions during process execution. This differs from the standard process execution that simply initiates the next task to be performed based on the task that has just completed. Evaluation of semantic effects allows a WfMS to decide which task is the most appropriate to initiate under the current conditions. Proposing suitable workflow management software is beyond the

scope of this document. However, one of the guiding principles behind this research is its potential use in WfMS.

2.4 Controlled Natural language

Natural Language refers to the common language spoken by people every day including the written version of that spoken language. Humans interpret Natural Language in a variety of ways based on understanding, context and personal opinion. There is always the potential for misinterpretation with any Natural Language which makes it unsuitable for machine interpretation. Machines communicate in structured languages such as formal logic. Statements in formal logic like the one below cannot be understood without training in the formal notation.

$$\exists x(Boxer(x) \longrightarrow Dog(x)) \tag{2.1}$$

While machines may be able to communicate and reason with this type of notation, the majority of humans would have difficulty understanding it never mind writing it. Business analysts in particular are generally not trained in writing formal notation. Since business analysts are a primary contributor to semantic effect annotation the notation employed must be within their current skill set if the framework for semantic effect annotation is to be adopted.

Natural Language would be an ideal medium for business analysts to annotate semantic effects to business process models. However, as previously discussed, they suffer from ambiguity and although great strides have been made in the field of Natural Language processing it was not considered to be mature enough, at the time of this publication, to warrant further testing. One alternative considered was Controlled Natural Language (CNL) [131] which is roughly positioned somewhere between Natural Language and formal logic with regard to its understandability. In earlier work [63] CNL was proposed for writing semantic effect annotations.

CNL is a subset of Natural Language and there are two varieties. The first type of CNL is a simple form that concentrates on eliminating ambiguity in documents like instruction manuals. The misinterpretation of written instructions in maintenance manuals could lead to catastrophic failures especially in the aerospace industry. Simplified Technical English was originally developed by the Aircraft European Contractors Manufacturers Association (AECMA) to avoid this type of misinterpretation. The AECMA was later subsumed by the Aerospace and Defence Industries Association of Europe (ASD) which currently maintains the standard (STE-100[50]). CNL's in this category are primarily designed for human consumption. As such they are well suited for machine-to-human communication because of

their characteristic ability to minimise misunderstanding. They are, however, not suitable for human-to-machine or machine-to-machine interaction.

The second type of CNL again is a subset of Natural Language but is based on formal logic. CNL's in this category can be easily interpreted by both humans and machines making them firm candidates for semantic effect annotation. Their qualifying features include that they have interpreters that translate between them and first order logic and they can be easily understood by humans that speak the natural language on which they are based. However, they still require the practitioner to be trained in how to use them correctly [133] thus placing an additional burden on the business analyst. Once trained though, the semantic effects annotated by a business analyst would appear, to the user, to be written in a natural language and could also be interpreted by a machine. Calculations derived from formal logic processing could also be translated into CNL for human interpretation [134]. CNL could therefore act as a bridging mechanism between human and machine with regard to the interpretation of textual content. Further information about CNL can be found at [7].

A CNL that is becoming popular within the semantic web community is Attempto Controlled English (ACE)[45, 77]. ACE has a comprehensive set of tools supporting its use for knowledge representation [44]. Semantic effect annotations written in ACE are human readable, translatable into FOL and thus can be reasoned with by machines. In the following observed effect, written in natural language, the business analyst has combined three separate effect statements into a single sentence using 'and' conjunctions.

> *The purchase amount is confirmed with a confirmation number and the client is the owner of the object purchased and the credit limit of the credit card is decreased by the purchase amount.*

It can be seen from the above verbose example just how varied semantic effect annotations can possibly be written without a strict syntax. When deconstructed it can be seen that there are three separate observations the business analyst intends to record.

> *The purchase amount is confirmed with a confirmation number.*
> *The client is the owner of the object purchased.*
> *The credit limit of the credit card is decreased by the purchase amount.*

The same sentences written in ACE would look like the following:

A purchase has a purchaseAmount that is confirmed by a confirmationNumber.
A purchase has an objectPurchased that is owned by a client.
A creditCard has a creditLimit that is decreased by a purchaseAmount.

The above sentences are as easy to understand as the original sentences written in natural language but there is an underlying structure to the way the ACE sentences are constructed allowing them to be converted into FOL. First the ACE sentences are paraphrased into the following:

There is a purchase X1.
There is a confirmationNumber X2.
The confirmationNumber X2 confirms a purchaseAmount X3.
The purchase X1 has the purchaseAmount X3.

There is a client X4.
The client X4 owns an objectPurchased X5.
The purchase X1 has the objectPurchased X5.

There is a creditCard X6.
The purchaseAmount X3 decreases a creditLimit X7.
The creditCard X6 has the creditLimit X7.

The paraphrasing can then be assembled into the following FOL statements that are machine processable:

has(purchase, purchaseAmount)
confirms(confirmationNumber, purchaseAmount)
owns(client, objectPurchased)
has(purchase, objectPurchased)
decreases(purchaseAmount, creditLimit)
has(creditCard, creditLimit)

Using CNL to represent semantic effects only solves the problem of translation to and from a formal language. Machines may be able to reason with the effects and present their conclusions in a human readable format but ambiguous terms can still be used in CNL without the use of a domain specific ontology. A domain specific ontology also facilitates the translation of CNL into First Order Logic (FOL). Languages such as OWL-S use variable names in their effect descriptions, e.g. *objectPurchased*. Such a term would not ordinarily exist within the taxonomy of a CNL application. A CNL interpreter (an application that can translate CNL into FOL)

would require a domain specific ontology that includes such terms. CNL provides a practitioner-accessible language for specifying effects (and thus, avoid the problems associated with insisting that analysts become proficient in a formal notation) while still making it relatively easy to translate these into an underlying formal representation. Its formal interchange capability and human readability positions ACE as a suitable candidate for the language underpinning semantic effect annotation. Its only drawback is that the business analyst must first be trained in its correct usage.

2.5 Ontology

The meaning and correct usage of the term 'ontology' is a hotly debated topic within the fields of computer and information science. Traditionally ontology is associated with a branch of philosophy known as metaphysics. It refers to the study of the nature of being and in this regard it is a science. Ontological study generates hierarchical graphs that categorise things and their relation to each other. The computing industry has adopted the term 'ontology' to represent the *knowledge graphs* generated by ontological study. A conflict exists between whether ontology is the scientific study or the product of that study. The term is now so widely used within the computing industry that its secondary meaning is now acknowledged within that domain. An ontology, with regard to computing, can be thought of as being a dictionary, thesaurus and relational database all rolled into one. The Oxford Dictionary defines an ontology as *"A set of concepts and categories in a subject area or domain that shows their properties and the relations between them"*. Some key words used in other definitions of *'ontology'* are, *'conditions'*, *'relations'* and *'dependency'* [135]. The key words refer to how an ontology links terminology.

The term 'boxer' could be used in a CNL sentence to describe a dog. It could also be used to describe a fighter. A machine cannot determine the intended meaning of the word without and ontology. When we write, "a boxer is a dog", the term 'Boxer' has an 'is a' relationship with the term 'dog'. It could also be described as being a subclass of the term 'dog'. However, this relationship only exists in the domain of 'pets'. In the domain of 'self defence', referred to earlier, a boxer is a subclass of something completely different. Likewise, an ontology for the packaging industry may link 'boxer' to an employee who packs boxes. The terms 'business' and 'company' have different definitions in a dictionary but they can be interchangeable in certain domains. In those domains they would be considered to have a 'same as' relationship.

Ontologies are domain specific but they can and often do refer to each other. Cross referencing literally improves an ontology. Take the case of the term 'boxer'; the pet domain ontology would include a subclass relationship between 'dog' and

'boxer'. A sentence in a piece of text may use the word 'boxer' and reference it to the pet domain ontology indicating that within the context of the sentence the word refers to a dog. The ontology becomes even more explicit when it cross references the self defence and packaging domain ontologies with links stating the term 'boxer' is 'different from' the term 'boxer' used in those other domains.

An ontology, as a relationship reference model of domain specific terminology, can ease the semantic annotation exercise in several ways (a substantial body of work explores the role of ontology in this setting). First, an ontology can help avoid *naming conflicts* in analyst-mediated immediate effect annotation (i.e., the same concept being referred to by different names). Second, an ontology can help resolve *abstraction conflicts*, where effect descriptions are provided at different levels of abstraction, and therefore in different vocabularies. Third, an ontology can provide the background knowledge base (KB) used for consistency checking of effect accumulation. Such a background KB can be easily obtained by extraction from an ontology those relations (rules) whose *concept signatures* are included in the concept signature of the effect annotations provided (i.e., that refer precisely to those concepts referred to in the effect annotations). Finally, an extension to the current tool is planned that leverages ontology to generate suggestions to analysts in terms of additional semantic effects that might be included in the annotation (for instance, a causal rule a ***causes*** b that might be implicit in an ontology might be used to suggest to an analyst that b be included in an annotation that contains a.

In [58], Guarino et.al. identify that *"An ontology formally specifies a domain structure under the limitation that its stakeholder understand the primitive terms in the appropriate way"*. Each of us has our own ontology that defines the way we interpret the world. Interpretation may not correspond with intended meaning causing confusion and misunderstanding. Knowledge Engineers craft an ontology to formally specify the meaning of the terms used in a particular domain. Everyone participating within that domain operates and communicates under those agreed meanings. Participants are said to have a shared ontological commitment [58]. This minimises confusion and facilitates cooperation.

An ontology defines the semantics but not the language. Although an ontology is a formal definition there are varying degrees of formalism. A basic ontology can simply be a definition of terms without any applied meaning whereas at the other end of the scale we find ontologies based on logical languages like full first-order logic (FOL). From the family of logical languages Description Logics are of particular interest because of their decidability and because they underpin many of the accepted standards for the development of the Semantic Web, covered later. Decidability refers to whether a decision can be reached, i.e., a deductive closure of reasoning. Tractability refers to how long a machine will take to arrive at a decision.

While FOL supports a high degree of expressivity, in many cases theorem proving with FOL can be intractable. Description Logics, on the other hand, guarantee deductive closure of reasoning with efficiency [102].

2.6 Description Logics

In 1977, KL-ONE [24] emerged out of Ronald Brachman's thesis [23] and is considered to be the first description logic. KL-ONE was developed in response to the lack of formal semantics in early semantic networks [143] and frame-based knowledge representation [105]. Description logics (DL) are decidable fragments of first-order logic [87] so they have a formal semantics. The expressiveness of description logics and predicate logics has been analysed in [21]. Researchers in the DL space are interested in how machines can efficiently make decisions. In the real world, applications need to make decisions within an efficient time frame, i.e., before the opportunity for action has passed. However, what has really brought DL out of the realms of academia and into mainstream popularity is its use in web-based applications [102].

Descriptions Logics are categorised based on the features they support. The features supported by a DL are represented by a standardised set of scripted letters, e.g. $\mathcal{ALC}$. The basic Description Logic is referred to as $\mathcal{ALC}$ which stands for "Attributive concept Language with Complements" [11], i.e., it is an Attributive Language that includes the complements operator ($\neg$). This basic language can be extended in which the extension of $\mathcal{ALC}$, that includes the transitive closure of roles [10], is abbreviated to $\mathcal{S}$. Extensions of different DL's include:

$\mathcal{H}$ - supports role hierarchy

$\mathcal{R}$ - supports role box

$\mathcal{O}$ - supports nominals/singleton classes

$\mathcal{I}$ - supports inverse roles

$\mathcal{N}$ - supports number restrictions

$\mathcal{Q}$ - supports qualified number restrictions

$\mathcal{F}$ - supports functional number restrictions

A Description Logic knowledge base can be decomposed into two component sections, a TBox and an ABox (see Fig:2.5). The TBox defines the terminology used within the domain of discourse, i.e., the concepts and roles (e.g. unary predicates in FOL). The ABox contains assertions constructed from concepts and roles defined in

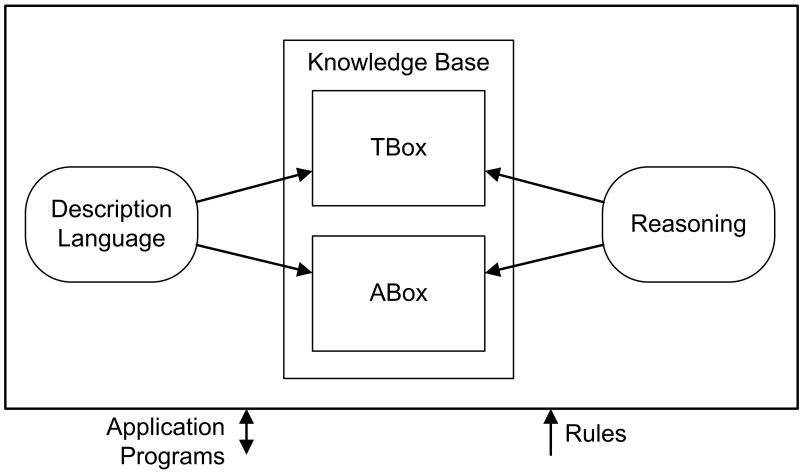

Figure 2.5: The architecture of a knowledge representation system based on Description Logics.[12]

the TBox (e.g. binary predicates in FOL). This breakdown of knowledge representation structurally underpins the framework described in this document. An *Effect*, as used in this document (see Chapter 3.2), is an assertion and the terms used in the construction of an *Effect* annotation must be defined within the TBox section of the background knowledge base. When a business analyst annotates an *Immediate Effect* to an activity they are free to use whatever terms they choose. This, in a sense, provides the flexibility of annotating in a natural language. Letting the business analyst choose their own words to describe an *Effect* removes the burden of learning a new language. This flexibility means that the application program used for semantic effect annotation must be able to contribute new terms into the TBox of the background knowledge base. Integrating those new terms is beyond the scope of a business analyst's training and so it must fall within the responsibility of knowledge engineers to perform those integration tasks.

The ABox of the background knowledge base will contain axioms that apply to all processes in a domain. Each *Effect Scenario* in a process is a knowledge base containing its own TBox and ABox. The terms defined in an *Effect Scenario* TBox ultimately must exist in the background knowledge base if accumulation is to be accurate. Accumulation is still possible even if the *Effect Scenario* terms do not exist in the background knowledge base TBox but the results of accumulation will not be reliable. Process models containing *Effects* that have not been integrated with the background knowledge base are flagged until knowledge engineers have integrated them. ABox is based on an open world semantics because in a real world environment it cannot be assumed the the knowledge in the knowledge base is complete.

2.7 Knowledge Representation

The explicitly defined relationships or links in an ontology eliminate any ambiguity surrounding a particular word. When a computer application processes textual content, it can determine whether the terms used refer to the content it is seeking. The terms used have a meaning which a machine can check by referring to an ontology. However, an ontology is only part of a knowledge representation system.

The annotation of semantic effects to business process models is an act of documenting knowledge. An expert, being interviewed by a business analyst, relates observed experience which is translated into a textual record and associated with a model element. If the text is unstructured, e.g. a plain text document written in a natural language, then machines cannot make sense of it never mind reason with it. If the knowledge is encoded in a machine language then humans cannot understand it without some type of interpreter. Before tackling the problem of reasoning with semantic effect knowledge we must first solve the problem of how that knowledge can be best represented both syntactically and semantically.

Research into Knowledge Representation (KR) is guided by five characteristic properties [35]. First, a KR is fundamentally a surrogate or substitution for the actual thing, i.e., the written word is a symbol used to represent the real world thing. Second, a KR is an ontological commitment, i.e., The thing that a symbol represents is dependent on the ontological commitment of the interpreter. Thus the same symbol can represent different real world things to different interpreters. A KR therefore establishes a fixed meaning to each word. All KR subscribers unanimously agree with the meanings. Third, a KR is a fragmentary theory of intelligent reasoning. The initial insight that led to the selection of a symbol to represent a thing is motivated by some act of intelligent reasoning. Typically, a symbol is only representational of part of the overall intelligent reasoning that led to the insight that motivated it. Fourth, a KR is a medium for efficient computation. Computational efficiency is of significant importance with regard to pragmatic application throughout the business community. Fifth, a KR is a medium of human expression. This final characteristic of a KR emphasises the importance of human-to-machine and machine-to-human communications and has been a driving principle behind this research.

This document refers to a background knowledge base with which *Effect Scenarios* (see chapter 3.4.1) must be consistent. The background knowledge base is part of what is referred to in [12] as a Knowledge Representation System (KRS) (see Fig:2.5). In [12], Baader et.al. depict a KRS as only contributing to application programs. This document explores the idea of application programs contributing to the KRS (see Fig:2.5) facilitating the evolution of a domain specific knowledge base.

2.8 Entity-Relationship Model

The Entity-Relationship Model [30] proposed by Peter Chen in 1976 included a diagrammatic technique for representing database design. The diagrammatic technique became popular because it could be intuitively understood by stakeholders with no prior technical knowledge of database design. The Entity-Relationship Model is based on the principle that the real world is made up of entities and relationships. Entities are things that exist in the real world and relationships are a type of referencing system that interconnects and defines entities. Chen goes on to identify the correlation between English language sentence structure and entity-relationship diagrams [29] in which he proposes eleven rules of translation.

2.9 Resource Description Framework

For a global exchange of knowledge to occur it is critical that semantic effect annotations comply with global standards. The Resource Description Framework (RDF) was originally created [89] as a simple data model with a standardised syntax for describing web related metadata, i.e., data about data. A *resource* within the original context of RDF refers to a Universal Resource Identifier (URI) that uniquely identifies where a *resource* can be found. The W3C first published RDF 1.0 [79] as a standard in 2004. It is often mistakenly thought of as an XML format but the RDF abstract data model has been encoded in many different formats of which XML is only one. The XML format is referred to as RDF/XML [94, 15]. The basic RDF model consists of three elements:

Resources - concepts, objects, things that exist

Properties - attributes of resources or relationships to other resources

Statements - a resource, a property and an attribute value for that property or;
- a resource, a property and another resource if the property expresses a relationship.

The extensible nature of XML has allowed RDF to evolve beyond its original scope where it is now commonly used to represent data. RDF Schema (RDFS) [59] extends the basic RDF model allowing it to represent vocabularies. RDF works on similar principles to the Entity Relationship Model. A sentence, written in English can be broken down into three component parts, a *subject*, a *predicate* and an *object*. These RDF statements are referred to as *"triples"*. In entity-relationship terms, the *subject* and *object* are entities and the *predicate* is the relationship between them. Simple sentences in a natural language can be decomposed into an RDF *triple*, e.g. *subject*(

The ball), *predicate*(**has the colour**), *object*(**red**). The RDF data model is used as a structured method for entering *Effect* data into semantic effect annotations. Throughout this document, whenever reference is made to something being written in RDF it refers to the structural nature of RDF (a *triple*) not RDF/XML format.

Further research [98, 20, 60] has shown how RDF/XML and RDFS can be extended to support knowledge representation. RDF has become a globally accepted W3C standard for knowledge representation on the web. In the early 2000's the Ontology Inference Layer (OIL) [72], also referred to as the Ontology Interchange Language, was proposed as an extension to RDFS [39, 26]. In 2002 Horrocks proposed a combination of the DARPA Agent Markup Language (DAML) and OIL to extend web markup languages, like RDF and RDFS, into a description logic. Effectively, RDF *triples* could be used to represent concepts defined in a description logic TBox. In 2003 Horrocks goes on to propose the Web Ontology Language (OWL)[71] which has now become another accepted W3C standard for the web.

2.10 The Web Ontology Language

The Web Ontology Language (OWL) [8] is a knowledge representation language. There are three varieties of OWL: OWL Full, OWL-DL and OWL Lite. OWL Full, like FOL, is so powerful that reasoning can be intractable making it inefficient for real world applications like those employed on the web. However, OWL Full is an extension of RDF which means that an RDF document is a legal OWL Full document and vice versa. There has been research into using an FOL theorem prover with OWL Full [147] but currently there has been no need for translating existing RDF documents into OWL so OWL Full falls outside the scope of this research.

OWL-DL [103] is equivalent to the description logic $\mathcal{SHOIN}(\mathbf{D})$ [70] where **D** indicates that some domain/datatypes have been integrated, e.g. boolean, integer, float and double. It is a sublanguage of OWL Full and is commonly used on the web because of its decidability and computational completeness, i.e., computations in OWL-DL are guaranteed to finish within a finite time. Any document written in OWL-DL is also a legal RDF document. However, not all RDF documents are a legal OWL-DL or OWL Lite document. Mappings from RDF to OWL and vice versa can be found in [118].

OWL Lite corresponds with the description logic $\mathcal{SHIF}(\mathbf{D})$ and was primarily designed for a market requiring only simple classification of terms. The DL extensions $\mathcal{SHIF}$ indicate that OWL Lite supports *role hierarchy*, *inverse roles* and *functional number restrictions*. It is a sublanguage of OWL-DL thus any OWL Lite document is a legal OWL-DL document. Like OWL-DL it is decidable but it has limited expressivity.

The latest version, OWL2 [13, 117], corresponds with the description logic $\mathcal{SROIQ}(\mathbf{D})$ meaning it supports the DL extensions *role box*, *nominal/singleton classes*, *inverse roles* and *quantified number restrictions* with some domain/data-types. As the OWL specification evolves we can expect further improvement in expressivity while still maintaining decidability.

An OWL knowledge base consists of *Classes* (DL Concepts), *Properties* (DL Roles), *Individuals* (Instances of Classes) and *Data Values*. There are two types of properties, Data Properties and Object Properties. A *Data Property* can be thought of as connecting an *Individual* with a *Data Value* (describing an attribute and its value) whereas an *Object Property* would connect an *Individual* to another *Individual* (a relationship). Each *Individual* is subsumed by a *Class* and *Classes* can subsume other *Classes*. These subsumption hierarchies are the type of models generated by the philosophical science of ontology.

The interconnection between FOL, DL, OWL, RDF and the English language can be expressed as follows. A FOL or DL binary predicate assertion,

```
role(concept, concept),
```

would translate into an OWL statement as either

```
object Property(individual,individual) or
data property(individual, data value).
```

These translations relate back to the RDF *triple* breakdown of an English sentence

```
predicate(subject, object).
```

This fundamental idea serves as a structural underpinning for the framework described in this document. One element of OWL that is used extensively in the implementation of the ProcessSEER framework is *Functional Properties*. A *functional property* can only have one value which makes it ideal for updating the values of *Effects* when reasoning about the outcomes of an activity. The tool, based on the ProcessSEER framework, utilises the OWL API [68] to generate OWL ontologies.

2.11 Semantic Web Services

The term 'Web Service' refers to the service offered by an application running on a web server. They are distinct from regular web applications first, because web services can exist without any graphical interface for manipulating them and second, because they can be controlled remotely. Web Service technology complies with many different standards like SOAP, WSDL and WSBPEL [152]. In their most basic form web services accept requests that may or may not contain data and return data as a response.

A Web Service is an application that offers an Application Programming Interface (API) and is described by a 'Web Services Description Language' (WSDL)

file [93]. That file usually resides on the web server where the Web Service application is located. A WSDL file, among other things, describes what information the Web Service requires as input and what information the Web Service will return as output. WSDL files are written in Extensible Markup Language (XML) and are therefore platform independent.

Another language associated with Web Services is the Web Services Business Process Execution Language (WSBPEL) [14] also written in XML format. WSBPEL is a scripting language used to coordinate Web Services and control how they interact. It works like an instruction set for a WfMS except that it is not restricted to in-house functionality potentially allowing WfMSs to communicate on a global scale. The standard is specified by the Oasis group, a consortium of public and private sector technology leaders, users and influencers involved in the development of open standards for the global exchange of information. There are other formats associated with Web Services but this thesis is only concerned with WSDL and WSBPEL.

Web Services are advertised in searchable repositories and one of the key concerns with this methodology is the ambiguity surrounding their description. Adding semantic annotations to Web Service descriptions has become an attractive proposition for overcoming this ambiguity, making discovery more precise. However, most Web Services currently only function on a syntactic level and lack the framework for semantic annotation [6].

Semantic Web Services (SWS) became a distinct research field in 2001 [97, 146, 4, 16] and has attracted a great deal of business interest but the implementation of SWS is still in its infancy. Because of the extensible nature of XML, the WSDL standard that describes a web service can be extended to include semantic annotations. In November 2005 IBM and the University of Georgia submitted a white paper to the World Wide Web Consortium (W3C) outlining a proposal for WSDL-S [5], an extension of the current WSDL specification. The document describes four fundamental semantic annotations of input, output, preconditions and effects for the purpose of improving service discovery. Other semantic annotation languages like Web Service Modelling Ontology (WSMO) [27, 84, 126] and Semantic Web Ontology Language (OWL-S) [96, 84] offer much richer expressivity for describing web services but they were regarded as being too revolutionary to be adopted by the WSDL using community. All these languages contain the same four fundamental annotations. These annotations are also necessary components for the automated analysis of business process models. They offer a framework for machines to make informed decisions about the services they wish to interact with. The potential therefore exists for business analysts to directly affect the orchestration of Semantic Web Services based on dynamic feedback from web service description files.

WSDL-S has evolved into the W3C recommended standard Semantically Annotated Web Services Description Language (SAWSDL) [129]. This new standard extends WSDL to include semantic annotations. Web services can consequently be composed using ontology reasoning rather than having the composition hard coded. This will allow a web service to seek out its own collaborating web service and negotiate independently. Rules and conditions can be applied to a web service that govern when and how it will enter into any transaction. As more web services utilise the SAWSDL standard so will the prevalence of automated transactions increase.

A key component of this research was to identify a mapping path from expressions and terminology familiar in the business domain to the more technical requirements of an application specification. XML is the key language underpinning all these other standards. Models in BPMN can be translated into WSBPEL, SAWSDL files can accept semantic effect annotations from a semantically annotated BPMN model if those annotations conform with an XML based standard like RDF and RDF effect annotations can be stored and reasoned with in an OWL ontology. There is one additional piece to the puzzle and that is to provide the ability for a web service to decide the type of web service it needs to transact with and that requires it to evaluate the current conditions and choose an appropriate collaborating service. This is different from one web service simply evaluating which of a selection of similar web services will be the most appropriate. The web service choreography may have to choose between a selection of completely different services so rather than a fixed choreography in WSBPEL an individual web service has the freedom to decide for itself where the execution path will lead. To do that involves Artificial Intelligence (AI) planning.

2.12 Action Languages

There are two types of Action Languages, *Action Description Languages* used to describe a transition system and *Action Query Languages* used to make assertions about a transition system[49]. The research presented in this document is primarily interested in *Action Description Languages* at this time. The Oxford Dictionary defines a *Language* as:

> *A system of symbols and rules for writing programs or algorithms.*

Expanding on this definition we define a *Language* as a formal *set of symbols* governed by a *set of rules* that determine how those symbols can be combined to infer meaning. Under this definition an Action Description Language is a set of symbols, representing actions, that are governed by a set of rules that determine how those actions interact, and their consequences. Fundamental to the set of symbols is their

underlying structure which Gelfond et al. [49] refer to as an *Action Signature*. An Action Signature is a template that contains all the functions and data needed to transform one state into another in a transition system. A state transition in its simplest form can be represented as $\langle s, A, s\prime \rangle$ where s is the original state, A is an action and $s\prime$ is the consequent state. An Action Description Language describes how the action A transforms s into $s\prime$.

The Stanford Research Institute Problem Solver(STRIPS) [40] is an AI planning tool that was introduced in 1971 for reasoning about actions. The STRIPS acronym has since been adopted to represent a formal language used to describe how actions are evoked and how they affect an environment. An *Action Description* for a STRIPS planner is a pair $\langle P, X \rangle$ where P and X are super sets of conditions. The set P contains two sets $\{Pt, Pf\}$ where Pt is the set of preconditions that must be true and Pf is a set of preconditions that must be false for the action to execute. The set X contains two sets $\{Xt, Xf\}$ where Xt is a set of conditions that are made true and Xf is a set of conditions that are made false as a consequence of the action being executed.

The Action Description Language (ADL) [119] extends the STRIPS language providing improved expressivity. An Action Description in ADL is a tuple $\langle P, A, D, U \rangle$ where P is the set of preconditions, A is a set of conditions to be added to the state description, D is a set of conditions that must be deleted from the state description and U is a set of clauses used to update the state description. ADL operates on an open world assumption whereas STRIPS operates on a closed world assumption.

The Planning Domain Definition Language (PDDL) [101] has evolved from a number of different AI planning languages of which ADL is one. The effort behind the development of PDDL is motivated by an attempt to standardise planning languages. PDDL separates a model of a planning problem into a domain description and a problem description making the problem easier to specify. The language employs the same information as other planning languages albeit in a different format. What is evident from these Action Languages is that they all require data in the form of actions, state descriptions, preconditions and effects. The functionality behind how an AI planner processes these data is beyond the scope of this body of research.

In this document, Action Description Languages are explored purely from a supplier perspective, i.e., providing the data needed for an AI planner to reason with and construct a plan from business process information. Providing the correct data in the correct format to a planner offers the potential to dynamically generate business process models. It is shown later in this document how the framework for semantic effect annotation of business process models provides all the elements necessary for an organisation to build a library of action descriptions that can be

utilised by an AI planner which in turn could generate dynamic instruction sets for a WfMS.

2.13 Possible Worlds Approach

The STRIPS approach to reasoning relies on the programmer including every propositional statement about the world that must be changed when an action occurs. The outcome is predefined and requires the programmer to know and record every possible consequence. If a rule in the background knowledge base states that a particular combination of propositions infers another proposition then the additional proposition will not be included in the consequent world description unless explicitly included into a STRIPS add list. The more complex the world description, the more onerous the task, of specifying add and delete lists for actions, becomes.

Ginsberg and Smith address this problem of inferred states with their Possible Worlds Approach [54]. Inferring states can lead to unexpected results that render a world description inconsistent. Ginsberg et al. overcome this problem by reviewing consistent subsets of the inconsistent world description. Subsets that are considered maximal (i.e., they represent a world closest to the original world with the minimum number of changes) are treated as possible worlds. Using this approach, inference rather than action can affect an outcome. The definitions provided in [54] are based on a world description that contains all formulae about the world including the rules governing it. These rules are referred to as protected sentences that cannot be changed. An example of a protected sentence given by Ginsberg et al. is:

$$on(x, y) \wedge y \neq z \rightarrow \neg on(x, z)$$

Translated, it states that an object x cannot be *on* two different things at the same time. A consequent world must include these protected sentences. An action's add list is a set of formulae that constitute a partial description of a consequent world. An action's delete list is no longer required when using this approach. If the union of the consequent world with the existing world is inconsistent then the maximal subsets of the unioned worlds become the possible worlds that could follow from the action. The maximal subsets must include all protected sentences and must include all formulae from the consequent partial world.

Definition 2.13.0.1. Let S be an existing state description of the world such that $S = \{s_1, \ldots, s_n\}$ where $s_i \in S$ is a well formed formula. Let R be a set of inference rules such that $R \subseteq S$. Let A be an action occurring in S. Let C be a partial consequent state description of the action A such that $C = \{c_1, \ldots, c_n\}$ where $c_i \in C$ is a well formed formula. Let W be a possible world that results from performing

the action A in the existing world S. To qualify as a possible world W must satisfy the following conditions:

1. W is consistent
2. $C \subseteq W$
3. $R \cap S \subseteq W$
4. $\nexists W\prime | W \subset W\prime$

The possible worlds approach is utilised in the ProcessSEER framework for reasoning about actions and is henceforth referred to as *Accumulation*. A business analyst need only annotate the partial consequent states to activities in a business process model. The delete list mentioned in many AI planners is not necessary and can be derived from the resulting possible worlds generated by a reasoning engine. This allows the tool to answer the question, "What would be the result if the process had executed up to this point?"

2.14 Clinical Process/Careflow Management

The notion of *careflow management* [115] has become the focus of considerable research attention. It builds on the premise that process management principles and techniques can deliver value in encoding (and coordinating execution via process engines) of clinical guidelines. More generally, careflow management also addresses the administrative aspects of health care, both from the perspective of health care providers and patients [34]. Clinical procedures are performed by a variety of clinicians and treatments are often prescribed autonomously. Even with the abundance of text-based documentation on medical procedure it can be difficult to identify potential conflicts between treatments. Patient records may contain a summary of existing and past treatments but lack certain details that could impact on future or concurrent treatments if undetected. Having this text-based documentation does not guarantee that it will be used whereas computer-based patient-specific reminders that are integrated into the clinician's work flow have proven to be far more effective [144].

Careflow processes are often represented in a diagrammatical format that provides a visually intuitive representation of the activities required to treat a patient's condition. Although process modelling has been used extensively in the business community it is a relatively new innovation within the health care industry. In [34] Curry et al describe a tool that utilises Workflow Reference Models [67] to visualise a patient's journey through a Health Care Organisation (HCO). The models

prove to be particularly useful for encouraging group communication within a HCO and promoting ownership and responsibility among active participants involved in the process. The models are particularly good for training purposes because they visually translate much easier than a text document.

The integration of Careflow Management Systems (CMS) can greatly improve a patient's journey through the health care system. CMSs can be used in a variety of ways, as control mechanisms to constrain the operations of health care workers to predefined treatment protocols [33], as decision support mechanisms that assist clinicians with prescribing treatment protocols [99] and as central repositories for the comparison and analysis of different treatment protocols [128]. Of these three only [128] addresses the need for internet compatibility so that stored information may be accessed by any HCO. This is particularly important given that a patient's journey through the health care system will place them in the hands of many autonomous HCOs. Internet access to other autonomous HCOs' treatment protocols would greatly assist clinicians with the prescription of their own treatments [9].

Clinical Decision Support Systems (CDSS) fall into six categories, 'Alerts and Reminders', 'Diagnostic Assistance', 'Therapy Critiquing and Planning', 'Prescribing Decision Support Systems', 'Information Retrieval' and 'Image Recognition and Interpretation' [32]. Relative to this taxonomy, the research reported here corresponds most closely to 'Prescribing Decision Support Systems', and to a lesser degree to 'Alerts and Reminders' and 'Therapy Critiquing and Planning'.

2.15 Artifact-centric Business Process Models

Artifact-centric business process modelling (ACBPM)[73] focuses on the artifacts that are affected by activities in a business process. Whereas planning notations like PDDL have at their core, a structure for specifying an action and its effect on the world, ACBPM focuses on a structure for specifying the artifacts and how they change. Principally, the conditions specified in a planning action describe the states achievable by artifacts in ACBPM. The difference lies in the way they are stored. Artifact descriptions are associated with the action in a planning notation whereas actions are associated with the artifact in ACBPM. In a very simplistic way these two approaches to modelling *change* in the world, bear the same relationship to that of an activity diagram and a state diagram in the Unified Modelling Language (UML). A point of interest in this research is the difference between these two perspectives and how that affects process modelling.

Cohn et al. [31] goes much deeper into the life-cycle of artifacts which gives them a life of their own, independent to the actions performed on them. The data in this case is related. The artifact-centric model uses the artifact as the container

for state descriptions that only apply to that artifact. It also contains rules governing allowable state descriptions and the order in which those state descriptions can be applied. Although some of these rules may be imposed by humans, many will express fundamental physical laws that apply to the artifact.

ACBPM is built on a four dimensional framework "BALSA", Business Artifacts, Life-cycles, Services and Associations [73]. Business Artifacts translate into concepts but the concept is only a label for a mini database of facts about the concept. Life-cycles would be a part of each concept's mini database that govern the order in which state descriptions can occur. Services refer to business process activities and Associations connect Artifacts to Services. In this document, an effect is regarded as a description of an Artifact and a Service is an activity. The act of annotating an effect to an activity establishes an Association but the notion of Life-cycles introduces an additional feature. Life-cycles denote an ordering of state descriptions for a single artifact or class of artifacts [156]. Life-cycle constraints can be used to question the observations of experts or the interpretations of business analysts during the design phase of business processes thus improving the quality of design.

2.16 Related Work

A lot of effort is currently being directed into semantic annotation for web service or process discovery. A semantic annotation framework was developed to facilitate the interchange of process models and their discovery [92]. Ontology is used in this framework as a classification repository for the identification of processes or subprocesses that satisfy the selection criteria. The tool specified in this document will reduce the risk of modifying existing processes by alerting the analyst to the consequences of design time decisions. We use ontology to define the vocabulary used in the *Effect* annotations such that they can be translated into formal logic. Our process differs from that described in [92] in that *Effect* annotations are not simply used for term comparison but also for reasoning about process outcomes.

A framework is described in [66] that proposes using a Status and Action Management model (SAM) for generating process models to achieve a specified goal. A SAM is referred to as a *business object* that closely resembles the artifacts described in [73, 31]. The business object is the focal point to which state descriptions and actions are linked. The method presumes that a fixed state description will always result from a particular action when performed on a business object. This research recognises that these fixed state descriptions (*Immediate Effects*) form only part of the overall outcome of an action. In fact some *Immediate Effects* may actually change when an action is performed on a business object under different circum-

stances. In [66] the repository of business objects is searched to devise a plan for reaching a goal, i.e., a process model. A plan is devised by assembling a sequence of actions such that the effects of each action satisfy the preconditions of the following action until an action is reached whose effects satisfy a goal condition. Hoffmann et al.[66] propose the complete replacement of existing state descriptions by the *Effects* of the action on the business object whereas this research recognises that the outcomes can be affected by the sequence of actions when *Effects* are allowed to propagate. The ProcessSEER framework can reveal side-effects that may not have been considered. The framework also utilises a business artifact repository but it is not the sole source of reasoning.

In [142], a Generic Process Model (GPM) is proposed to encode and extend the representation of processes with state and stability (i.e. goal) relevant information. These notions of state and stability lead to a general notion of validity of process models (primarily w.r.t. goal reachability). In [141], the GPM is used as a basis for identifying the scope of changes that can be made to an existing process given changes to GPM-related phenomena (e.g. goal change). Some of the techniques outlined in this paper, such as the accumulation procedure, help leverage partial and symbolic state descriptions to perform goal and change relevant analysis. In the SBPV approach [150], a scheme for annotating and propagating a restricted form of axiomatic task descriptions is introduced for a restricted class of process models, but differs in several key ways to this work. The approach presented in this document provides a parsimonious extension to the modelling framework (the analyst's effort is only extended by requiring *Immediate Effect* specifications of tasks in the BPMN model) and is driven by the need to identify the minimal amount of semantic annotation required to meet the requirements of functions such as compliance management, process change and life-cycle management, enterprise process architectures etc. The SBPV approach, on the other hand, requires complete specifications of both pre-conditions and post-conditions that are context-sensitive, thus placing a somewhat onerous burden on the analyst (besides additional annotations that are required for reachability analysis, which are not considered in this framework). The machinery for contextualising context-independent task *Effect* specifications, provided by analysts, solves a harder problem, by permitting non-determinism in *Effect Scenarios*. Consequently the ProcessSEER framework cannot provide polynomial-time guarantees as the SBPV framework can. This is not considered to be a significant impediment since design, annotation and propagation tasks do not normally involve real-time constraints, and afford the luxury of slower off-line computation. Evaluation shows that the ProcessSEER framework is still able to meet reasonable processing-time bounds. In [55], similar process annotation techniques are used for compliance checking.

Many clinicians are hesitant to adopt Workflow Management Systems (WfMS) because of the prescribed approach to performing activities. The user must follow the procedure to the letter. In a clinical environment this is rarely the case. Procedures need to be modified continuously to cater to the ever changing circumstances. A WfMS that offers no guidance until the prescribed task has been completed becomes useless when a clinician deviates from the prescribed path of operation. Mulyar et al. seek to overcome this problem with their CIGDec language [107] that provides flexibility in the specification of workflow models. The flexibility allows a WfMS to recover from deviant behaviour by the user. Guidance can still be offered to the user through constraints applied to certain activities, e.g., this activity must follow this activity. If no constraints exist then the clinician is free to choose from a selection of activities.

The CIGDec approach relies on constraints to provide limited control flow. The constraints are based on pre- and post- conditions. In the examples provided, those conditions refer to activities, e.g., a blood test cannot be enacted until the patient has consulted with a doctor. The WfMS is still functioning on an imperative framework. Decisions are based on actions and guidance is offered on the dependencies of actions to each other. The activity dependencies have no connection to patient related data. However, the ability to specify whether a sequence flow is a hard or soft constraint has definite application to this research. This capability would assist in the selection of next activities to perform. The BPMN 2.0.2 Specification [111] does not cater for these hard and soft constraints. All sequence flows can either be interpreted as hard constraints, leading to an inflexible WfMS, or all soft constraints, leading to a completely flexible WfMS that simply presents the user with a set of activities from which to select.

The ProcessSEER framework offers a completely flexible approach but provides guidance about activity selection based on patient state descriptions. The back end rules govern state transitions not activity transitions. Activities are selected based on desired outcomes, not on a rule governing the prescribed order of execution. There is scope for applying both types of rules in a WfMS but the potential exists for conflicts to occur between the rule types. If a patient condition dictates that a particular activity should be enacted and an execution rule dictates that another activity must be enacted immediately given that some activity has just been completed then the two prescribed activities could potentially be in conflict. Combining both sets of rules could be a challenge.

Of particular interest for this research is the patient state step in GLIF3 [22] that allows the state of a patient to be encoded. The encoding then affects how the WfMS compensates for deviant behaviour. The WfMS must be able to interpret the encoded state which therefore must comply with the terminological rules of a

back end ontology. The ProcessSEER framework provides this functionality and the prototype tool complies with the RDF standard. The potential exists for the data generated by the ProcessSEER framework to be integrated with computer-interpretable guideline languages that could act as a flexible instruction set for a completely dynamic WfMS that could assemble workflow activities in real time. Although the details behind the instruction sets governing a WfMS are beyond the scope of this research, the work herein has a definite application with regard to languages that govern WfMS operation.

2.17 Summary

This chapter discusses the different technologies needed to support a framework for semantic effect annotation of business process models. The diverse ways in which the ProcessSEER framework can be utilised are also explored, e.g. semantic web services and automated planning. Clinical use cases in careflow management are also discussed. The chapter concludes with a review of similar work.

Chapter 3

The ProcessSEER Framework

ProcessSEER stands for Process Semantic Effect Evaluation and Reasoning. The framework provides a means by which a business analyst can annotate business process models with semantic effects that support automated reasoning and evaluation of model design. Before embarking upon a definition of a framework for semantic effect annotation of business process models it is necessary to clearly understand the two fundamental elements in a business process that contribute to this framework, actions and effects.

3.1 What is an Action?

The Oxford Dictionary defines an action as *"A thing done; an act"* or *"A gesture or movement"*. As a noun, an action becomes a thing, a concept that is objectified in our minds. In reality, the only thing about an action that is objectifiable is its name, the words used to describe the action. When we refer to a thing performing an action we are referring to a sequence of instances involving the thing being observed. This can be easily seen when we take video footage of a thing performing an action. The result is a series of instances or frames (see Fig:3.1) in which the observer (the camera) captures the position of the thing at a particular point in time. Using time-lapse photography we could capture a different series of instances. Some may match instances in the video footage while others may be unique. This is due to the different capture rates. We could capture one instance of a thing prior to it performing an action and another instance upon completion of the action and declare that in between these two instances an action occurred. In this case the action is the gap between two instances (frames). However, when we name an action we are typically referring to a sequence of observable instances. The action name is chosen to represent the sequence of instances not the gap between instances. The sequence is important, thus an action is representative of order. We therefore have two ways in which to consider an action, as a gap between states or as a sequence

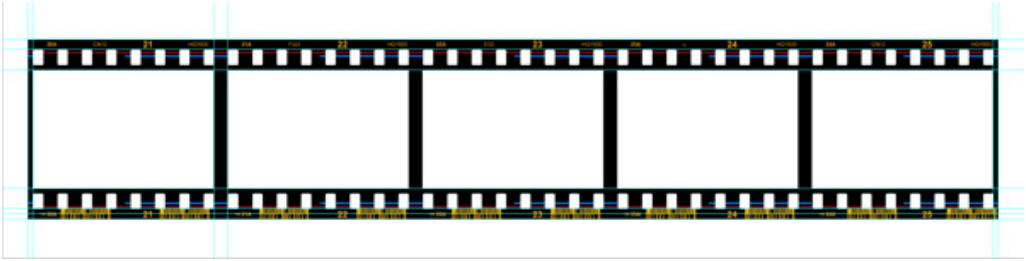

Figure 3.1: A film strip showing individual frames which are snapshots of the world from a specific point of view.

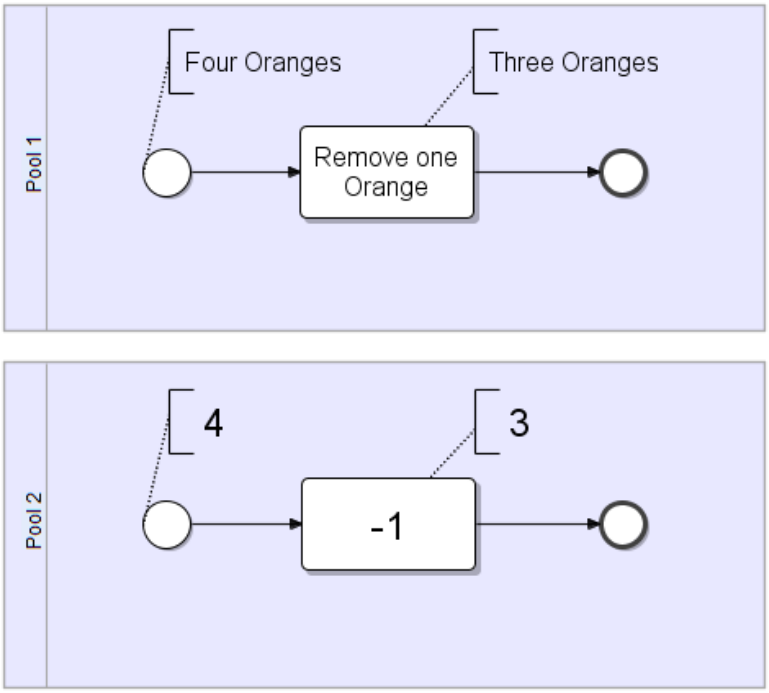

Figure 3.2: The two basic business process models contain text annotations indicating values assigned to the events and tasks. Events are indicated by circles, and tasks by boxes with rounded corners.

of states. Analysis of the sequences of states is commonly used in sport to improve performance and minimise injury. In business an action is more commonly viewed as the gap between one state and another, i.e., given the current state, an action is performed in order to achieve a desired state or in other words have a desired effect. We can also consider an action from a mathematical perspective. The two process models in 3.2 show the similarity between a process and an equation. It can clearly be seen in the bottom diagram that the activity is equivalent to a function, e.g. $-1(4) = 3$. The -1 function has a domain of 4 and a range of 3. The state descriptions (4 and 3) represent the pre and post conditions respectively. In this particular equation the -1 function accepts only integers and produces integers as an output.

An action written in an Action Language like STRIPS [40], ADL [119], PDDL [101] or OCL [149, 113] contains information about the existing state of the world, the consequent state of the world and an instruction set that controls the changes between the two states. Consider a state transition defined as $\langle s, A, s\prime \rangle$ where s is the original state of the world, A is an action performed within that environment and $s\prime$ is the consequent state of the world after the action has been performed. The action A in this case can similarly be considered as a function $A(s) = s\prime$ where s is the domain and $s\prime$ is the range of the function. Both s and $s\prime$ are sets of literals whose format will be discussed later.

The action A can also be viewed as a tuple (P, A, +L, -L) where P represents the preconditions, A is the action label, $+L$ is the add list (containing all propositional statements to be added to s) and $-L$ is the delete list (containing all propositional statements to be deleted from s). The action label is simply an identifier. Data that was previously accessed internally with $A(s)$ is now passed as parameters to the function $A(s, P, +L, -L) = s\prime$.

Artifact-centric process modelling [73] views actions (services) as transformations to a specific artifact. The artifact is the container for all the state descriptions and instruction sets for performing the transformations. Life-cycles are artifact specific rules that define sequential transformations of the artifact and relate those transformations to actions. An actionable function in artifact-centric modelling may look like $a(s_i, A) = s\prime_i$ where a is an artifact, $s_i \in s$ is a description of an existing state of the artifact, A is the action and $s\prime_i$ is the consequent state of the artifact. Life-cycle rules contained within the artifact along with the corresponding action dictate $s\prime$. Life-cycle rules pertaining to an artifact are changeable within the context of a process. Their applicability is governed by time and state descriptions.

In a BPMN process model an action is a textual description, inside a round cornered box (see Fig:3.2). An action can also be an event, represented by a circular icon. Actions are either performed by an agent or are enacted upon an agent. The description serves as a label to identify the activity. This label can also be used as an instruction but to do so imposes certain requirements on how the label is composed. An instruction must be an imperative sentence yet it was found that even within the BPMN documentation tasks were sometimes written as declarative sentences. These findings about task label composition are discussed later in chapter 6.

At a very basic level an action can be nothing more than an edge connecting two nodes in a directed graph. In all these representations of actions we find that an action, whether a function or a parameter, is nothing more than a label used to symbolise change. The label, whether it be a single word like *"move"* or a complete sentence like *"Hit the ball."* is a symbol that has been chosen to represent an observed change in the world. An Action Language seeks to take those labels and

translate them into actionable and decidable code that machines can utilise. An action is therefore defined as a label in the form of an imperative sentence that symbolises the transformation of one state into another.

Note that an action is not considered to be the thing that performs the transformation. The imperative sentence is an instruction to an agent or actor to perform the transformation. Under this definition an action does nothing as the name would imply. An action is nothing more than a symbol to represent change. Throughout this document the words *action* or *activity* are used as generic terms to represent a BPMN task or event.

3.2 What is an Effect?

If we are to develop a framework for semantic effect annotation of business process models then we need to be clear about what is meant by the term *'Effect'*. We asked a group of people, "What would be the effect of them chopping down a tree?" The answers varied but essentially each one described the effect as "the tree falls down". It is very common to think of effects in this way, I chop down a tree and the effect is that the tree falls down. In this case we are thinking of an effect as an action that immediately follows another action. Perhaps it is because we are thinking of the action performed by the tree rather than being performed by us that we regard the action of the tree as an effect of our own action.

We can also consider the effect of an action as being the overall effect on many different things, as with the damage a falling tree causes, or we can think of an effect as being in relation to only a single thing as with the tree. The word *"thing"* is used throughout this document to refer to either an object or a concept. In this case an effect, being a state description, can consist of just a single sentence or a collection of sentences such as a paragraph or even an entire document of text depending on the number of things being described. Given that an agent can also *effect* change in many things ("effect" used as a verb) illustrates the ambiguity surrounding the word "effect". To avoid this ambiguity the meaning of the word "effect" is further refined to only include a single sentence describing the state of a single thing that has changed as a consequence of some action.

The Oxford English Dictionary defines the word "consequence" as meaning "a result or effect" so we could also consider the tree falling down as a consequence of our chopping. The Merriam-Webster Dictionary defines "consequence" as "something that happens as a result of a particular action or set of conditions". The consequence in this case is something that happens indicating an action that sequentially follows another action or set of conditions. The fact that this consequential action is defined as a result qualifies it as an effect. According to the Cambridge

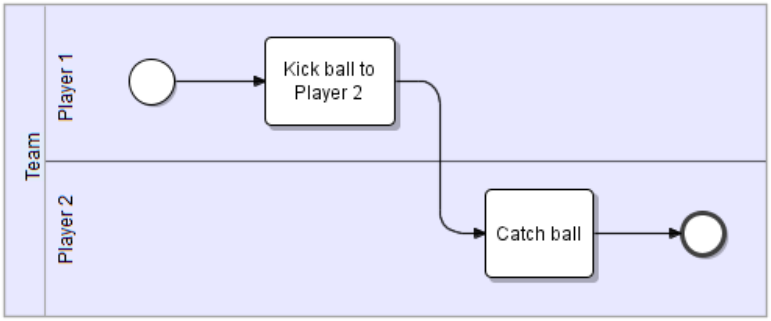

Figure 3.3: A basic BPMN model of two rugby players kicking a ball. (version 1)

English Dictionary the prefix "con" means "with" or "together". The word "consequence" would therefore mean "in sequence with", i.e. an action that follows sequentially after another action. The Merriam-Webster Dictionary defines "effect" as "a change that results when something is done or happens: an event, condition, or state of affairs that is produced by a cause." In all these definitions it can be clearly seen that an effect can be a statement describing either a static or dynamic state provided that the said state is the result of a particular action or set of conditions. The following examples explore how dynamic Effects can be represented in BPMN.

Using an incident from a game of rugby as an example we observe that Player 1 kicks the ball to Player 2 and Player 2 catches the ball. If we were to model this using BPMN it would look similar to Fig.3.3. Now if we think about what happens as a consequence of the first activity, i.e., Player 1 Kicks the ball to Player 2, we could be forgiven for thinking that 'The ball travels through the air toward Player 2' is the effect of this activity, but let's rethink this in the context of a BPMN model. A model represents actors as swimlanes, 'Player 1' and 'Player 2', and actions as activities or events, 'Kick ball to Player 2' and 'Catch Ball'. These are standardised symbols of the notation. The important thing to recognise is that actions in BPMN have clearly defined symbols. Now when we think about the statement, 'The ball travels through the air towards Player 2', we can see that it describes an action, an action of the ball. We could model this action as an event but then we are faced with the challenge of which swimlane to place the event since it is an event that happens to both Player 1 and Player 2 (see Figs.3.4 and 3.5). An alternative solution would be to treat the ball as an actor. Although not an instigator of activity, a ball still qualifies as an actor within the BPMN syntax. It can therefore be represented as a swimlane in which its actions are represented as activities (see Fig.3.6). We have made the claim that BPMN has no representation for the effects of activities yet clearly here we can see that an effect can be represented using the existing notation. The problem lies in the ambiguity surrounding the word, 'effect'. Consider the following alternative effect descriptions: "The ball is in the air"; "The ball is in the

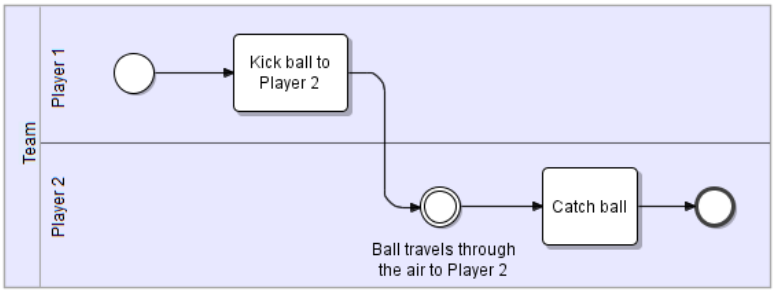

Figure 3.4: A basic BPMN model of two rugby players kicking a ball. (version 2)

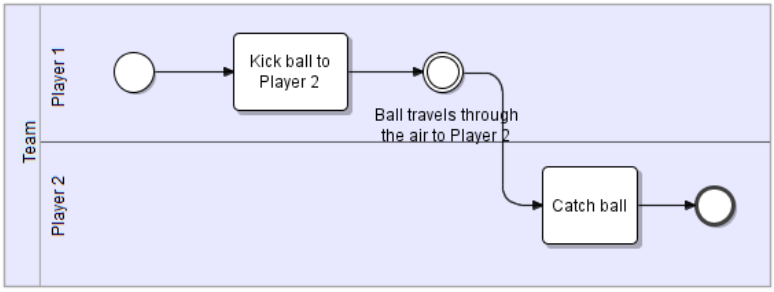

Figure 3.5: A basic BPMN model of two rugby players kicking a ball. (version 3)

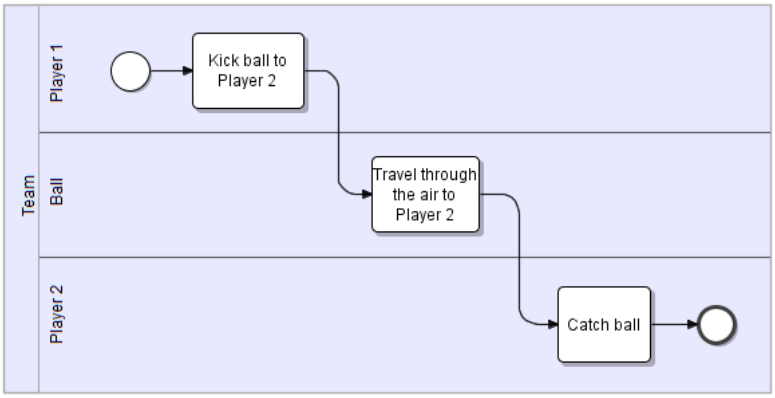

Figure 3.6: A basic BPMN model of two rugby players kicking a ball. (version 4)

hands of Player 2". In both these statements we are describing the state of the ball rather than the action it is undergoing. The thing that is missing from BPMN is the ability to represent the state of things and how they change as a consequence of actions. Thus, when we refer to the effect of an activity in a BPMN model we are referring to a state description of the world immediately following that activity. The state description can be abstract as in "the players are in their starting positions" or specific as in "Player 2 is in the Wing position" and "Player 1 is in the Fullback position" etc.

A dynamic state is one that is changing and implies motion or action. A sentence used to describe a dynamic state is an abstract description of a sequence of static states. If we consider the tree that is falling down and we film that dynamic state, we would get a sequence of individual snapshots (frames - static states). The effect, "The tree falls down", is a summary of that sequence of static state descriptions which for all intents and purposes is a far more efficient method of communication than to describe each static state in sequence. This idea follows from the previous section "What is an Action?".

The Business Process Modelling Notation (BPMN) provides task and event symbols for representing actions (motion sequences). A BPMN model could represent the effect of the tree falling down as a sequential activity occurring in a swimlane representing the tree. The preceding activity, "chop down tree", would be positioned in a separate swimlane representing the person doing the chopping. If the intention is to capture effects that include dynamic state descriptions then the ProcessSEER effect annotation framework will overlap with the current BPMN framework. The purpose of effect annotation is to explicitly capture information that would otherwise be implicitly represented in BPMN. To achieve this, effect statements must be limited to static state descriptions.

While we may not be interested in every static state that occurs during a motion sequence, we are interested in the static states immediately prior to and following that sequence. These changes in state are referred to as state transitions [49]. Each state transition contains an existing original state description and a consequent state description and refers to how the state of the world changes as a consequence of some action. The consequent state description is often referred to as the effects of an action. Note the deliberate use of the plural, effects. A state description is therefore a collection of one or more declarative sentences describing the static states of one or more things and by definition, an effect is one of those declarative sentences within a state description. Effect sentences may each describe different things or they may describe different aspects of the same thing. A sentence describing the state of a single thing can either be describing an attribute of that thing or its relationship to some other thing. Thus a state description of a single thing can contain many

declarative sentences describing its different attributes and its different relationships to other things.

A single declarative sentence describing the state of a thing is still not a refined enough definition to be workable. A single sentence can contain descriptions of one or more attributes and or relationships between things, e.g. the water is boiling in the pot on top of the stove. Such a declarative sentence can be broken down into three separate sentences:

1. The water is boiling.
2. The water is in the pot.
3. The pot is on top of the stove.

Sentence 1 describes an attribute of the water while sentences 2 and 3 describe relationships between the water with the pot and the pot with the stove respectively. An effect is therefore differentiated from a declarative sentence by constraining it to only describe a single attribute of a thing or a single relationship between one thing and another. Breaking down each of these sentences into their component parts, subject, predicate and object, reveals:

1. The subject is always the thing being described.
2. The predicate is either an attribute or a relationship.
3. The object can differ depending on whether the sentence is describing an attribute or a relationship. A sentence describing a relationship will always have another thing as its object but the object of a sentence describing an attribute can be either a value or another thing. For example, "The ball has the colour, red" has a subject, "The ball", a predicate, "has the colour" and an object, "red". The colour "red" is a concept not a value whereas the sentence "The ball has a diameter in centimetres of 15" uses the value "15" as its object.

Decomposing a sentence into its component parts provides a framework for the manual annotation of effects. However, those component parts can contain different types of information. This differentiation is explored further in the section on Manual Annotation but for now, a sentence describing an effect can be translated into a binary predicate statement in the following forms:

- $Attribute(Subject, Value)$
- $Attribute(Subject, Object)$
- $Relationship(Subject, Object)$

Rule 11 in [29] states:

> *"a sentence in English corresponds to one or more entity types connected by a relationship type, in which each entity type can be decomposed (recursively) into low-level entity types interconnected by relationship types".*

The premise behind this rule is that sentences can be decomposed into clauses which can be further decomposed into subclauses and the corresponding entity types can be decomposed into, what Chen refers to as, low-level entity types with interconnecting relationships. Sentences that would otherwise be represented by n-ary predicates can therefore be decomposed into a conjunction of binary predicates.
The three *Effect* binary predicate examples above consist of five component parts:

- *Subjects*
- *Objects*
- *Values*
- *Attributes*
- *Relationships*

These five categories can be further reduced into:

- *Concepts* and
- *Relations* (roles in FOL)

Under this categorisation *Concepts* subsume *Subjects*, *Objects* and *Values* and, *Relations* subsume *Attributes* and *Relationships*. An *Effect* is therefore defined as a single sentence that describes the static state, of a relationship or an attribute, of some *thing* that has changed as a consequence of some action.
An *Effect* is either a positive binary predicate statement
$Relation(Concept1, Concept2)$
or a negative binary predicate statement
$\neg Relation(Concept1, Concept2)$
where *Relation* is the predicate of the sentence, $Concept1$ is the subject of the sentence and $Concept2$ is the object of the sentence. The positive and negative versions can be interpreted respectively as the *Effect* holds or does not hold in the current state description. In this regard an *Effect* is an ontological assertion.

This basic syntactic structure, $Relation(Concept1, Concept2)$, underpins all *Effect* annotations referred to in this document. Effects are annotated to BPMN tasks and events and grouped into what is referred to as an *Effect Scenario* (see 3.4.1).

3.3 What is a Condition?

Dictionary definitions describe an effect as the result of an action or set of *conditions*. Again, there is ambiguity surrounding the word "Condition". The difference between an *Effect* and a *Condition* is semantic rather than syntactic. The difference lies in how the statements are interpreted. An *Effect* is a statement of fact whereas a *Condition*, in the context of this research, is a query used to test for the existence of *Effects*. Gelfond and Lifschitz [49] refer to *Conditions*, and specifically preconditions, as fluents, variables to which no truth value has been assigned whereas *Effects* are literals, having been assigned values of true or false. A condition can be assigned the value true or false under the notion of it being a fluent. However, within the context of this research, a Condition specifies its value by being represented as a positive or negative literal. A *Condition* is therefore defined as a single sentence that describes the static state, of a relationship or an attribute, of some *thing* and indicates that the sentence must exist in the current state description. A *Condition* is either a positive binary predicate statement
$Relation(Concept1, Concept2)$
or a negated binary predicate statement
$\neg Relation(Concept1, Concept2)$
where *Relation* is the predicate of the sentence, $Concept1$ is the subject of the sentence and $Concept2$ is the object of the sentence. The positive and negative versions can be interpreted respectively as the *Condition* must hold or must not hold in the current state description. In practice this translates into whether the corresponding *Effect* statement exists in the current state description. A statement that is missing from the current state description does not qualify as satisfying a negated *Condition*. A negated binary predicate statement specified by the *Condition* must explicitly exist in the current state description to satisfy the requirements of the *Condition*.

Within the BPMN syntax, *Effects* are annotated to tasks and events while *Conditions* are annotated to the outgoing edges of exclusive and inclusive gateway splits. Throughout this document *Effects* and *Conditions* are referred to as binary predicates. They are the base elements of the semantic effect annotation framework.

Preconditions

The Immediate Effects of an activity are indicative of the observed changes in the state of the world as a consequence of the activity. As such, an Immediate Effect is an assertion of fact that automatically over-rides any previously described conflicting state descriptions. When considering the effects of an activity, a business analyst must also consider the preconditions of the following activity. Do the Immediate

Effects of the current activity satisfy the requirements of the next activity? This may not always be the case. An Immediate Effect Scenario is only a partial state description that captures only those things that change as a consequence of the action. Conditions may already exist that were established prior to the current activity and providing the Immediate Effects of the current activity do not over-ride them then they will remain in effect for the next activity.

This establishes a previous Cumulative Effect Scenario as a legitimate checklist against which an activity can determine whether its execution requirements have been met. Much the same as an Effect can double as a gateway Condition (structurally they are identical) so too can an Effect be utilised as a precondition. This idea corresponds with the Petri net concept of Tokens. When a Token (Immediate Effect) is generated by a Transition it becomes the precondition for the next Transition.

The Cumulative Effect Scenario may contain extraneous Effects that are not absolutely necessary for execution of the next activity but that is the nature of preconditions anyway as identified in the field of AI as the "Qualification Problem". The Qualification Problem recognises the impossibility of listing all the preconditions necessary for an action to achieve its intended effects. The important thing to note is that a Cumulative Effect Scenario, even though only a partial state description, is saying, "This is the state of the world prior to the execution of the next activity". Every Effect statement therefore becomes a check point. If an Effect is missing or is different from what is expected then this is enough reason to question the continuation of process execution.

If Cumulative Effect Scenarios are treated as pre-conditional statements then multiple Effect Scenarios introduce an interesting process design check point. Given two alternative Cumulative Effect Scenarios, do both describe a world that is necessary for the next activity to execute? Both scenarios may very well describe states of the world in which the next activity can successfully execute which means that the Effects they share in common are the important preconditions necessary for execution. Alternatively, if one of the Cumulative Effect Scenarios describes a world that is not conducive to process execution then it indicates to the business analyst that a gateway is required in the design.

It is the business analyst's responsibility to make sure that the preconditions of each activity are met but rather than double the work required by mandating precondition annotations, they need only check that the necessary preconditions exist in the Cumulative Effect Scenario. If the preconditions do not exist then additional activities may need to be introduced that have the required Immediate Effects to provide those needed conditions.

If a business analyst is required to annotate preconditions to an activity then it is assumed that they know what those preconditions are. If they know what

the preconditions are then they can design tasks that will generate the required Effects necessary for the execution of the next task. The accumulation process will determine whether the required state descriptions will be in effect prior to the commencement of the next task.

The next question is whether it is necessary to define the state of everything? Can we use a closed world assumption for each process? It makes sense that we can because of the focused nature of a business process. It is only concerned with the objects and activities involved in the process. The same qualification problem arises when we consider exception handling in process models. We can never define enough exceptions to cover all contingencies. To do so would render the process model unreadable. This raises another interesting issue with regard to what we are trying to capture in a process model. In general it would appear that a process model is meant to describe the process by which a particular goal can be achieved given that no unexpected outside interference is encountered. Once we begin to cover the possibility of outside interference we run into the qualification problem again.

3.4 Scenarios

An observer's focus of attention is commonly drawn to state transitions in the observable environment being the consequences of actions performed in that environment. The state of everything else that does not change state, does not capture the observer's attention and thus is not recorded. A state description of effects will, in the majority of cases, be incomplete with regard to the overall environment. The focus of the observer's attention will limit the description to only things of interest while ignoring irrelevant things or it will describe only a limited number of attributes of a thing or its relationships to other things. When we observe the effects of some action we are selectively identifying the things that change while ignoring the things that are not of interest. For this reason, the effects described by a domain expert will only ever represent a partial state description within the domain of discourse. These partial state descriptions contain either one or many effects.

3.4.1 Effect Scenarios

State descriptions, in general, can contain one or many statements. They can be partial or complete. The statements used to describe an *Effect* are no different to those used to describe a *Condition* and are no different to those used to describe an observed state. What differentiates an *Effect* from an observed state description is its association with an action. Remove that association and the *Effect* becomes a

state description even though only partial. A state description may refer to a thing that has remained unchanged by an action. Recording these static states is referred to, in AI, as the *frame problem* [125] which inflates the amount of data, needed to accurately describe a state, thus increasing the computational cost of calculating state changes. For this reason an *Effect Scenario* is only a partial state description containing only descriptions of things that have changed as a consequence of some action or actions. *Effect Scenarios* are either annotated to activities in a BPMN model or are generated by an accumulation function, discussed in the Accumulation section 3.6. A state description containing only *Effects* is referred to as an *Effect Scenario*. Effect Scenario is a generic term used to describe a set of Effects. There are two types of Effect Scenarios, immediate (see 3.4.3) and cumulative (see 3.4.4).

Definition 3.4.1.1. Effect Scenario
An *Effect Scenario* is a set of *Effects* that can be associated with one or more actions. In this regard an *Effect Scenario* is a generic term for any collection of *Effects* where each *Effect* may or may not be associated with an immediate action performed in the current state. In other words an *Effect Scenario* may contain *Effects* from actions that have previously occurred.
Let $\langle T_1, T_2 \rangle$ be a sequence of two actions. Let ES be an *Effect Scenario* such that ES contains known *Effects* that exist after the execution of T_2. Let $(e, T_j)_i$ be an *Effect* such that e is associated with the action T_j and i is an index in ES. An *Effect Scenario* can therefore be represented as a set $\{(e, T_j)_1, (e, T_j)_2, \ldots, (e, T_j)_n\}$ where $j = 1$ or $j = 2$.
An *Effect Scenario* may also be referred to as a conjunction of *Effects* such that $ES = ((e, T_j)_1 \wedge (e, T_j)_2 \wedge \ldots \wedge (e, T_j)_n)$ where $j = 1$ or $j = 2$.
The Semantic Effect Annotation Framework utilises a background knowledge base (KBR), the specifics of which are discussed in the chapter 6. All *Effects* within an *Effect Scenario* must be consistent with the KBR, i.e., no *Effect* can contradict directly or indirectly any other *Effect* within the same *Effect Scenario* such that $ES \cup KBR$ is consistent.

This definition of an *Effect Scenario* is purely theoretical. When implemented, an *Effect Scenario* is much more complex containing instruction sets similar to those used in Action Languages for controlling change. The definition of an *Effect Scenario* is revised in chapter 6.

3.4.2 Condition Scenarios

When a person makes a decision to take a particular course of action then the world changes as a consequence of that decision. That means that an activity can

generate alternate state descriptions (Effect Scenarios) depending on the decision. For example, an activity, "Flip a coin", would generate two state descriptions, one in which the coin displayed heads facing up and the other in which the coin displayed tails facing up. An activity of this type may be followed by a decision gateway that splits the workflow. If the outcome of the coin was heads then workflow would proceed in one direction and if tails it would proceed in the other direction.

A *sequence flow* is a directed line that connects elements in a BPMN model. It indicates the sequence in which activities are executed in a process. It can also be referred to as a *path*, an *edge*, a *sequence edge* or a *branch*. A *sequence flow* connects the "Flip a coin" activity to an *exclusive gateway split* which splits the workflow into two outgoing *edges*. The semantics of a BPMN *exclusive gateway split* indicate that it controls along which of these outgoing edges the workflow will proceed. This would indicate that the *exclusive gateway split* generates a different *Effect Scenario* for each outgoing branch. If a decision gateway is treated as an action that generates alternative *Effect Scenarios* then it becomes difficult to specify which *Effect Scenario* applies to which outgoing edge. It could be argued that a decision does nothing and thus has no effect on the physical world.

The following method involves annotating alternate *Effect Scenarios* to the "Flip a coin" activity and *Condition Scenarios* to the outgoing *edges* of the *exclusive gateway split*. The alternate *Effect Scenarios* pass through the *exclusive gateway split* and are duplicated for each outgoing *edge*, i.e., two Effect Scenarios are passed along each *branch*, one containing the "Heads" *Effect* and the other containing the "Tails" *Effect*. The *Condition Scenario* on one *branch* will contain the "Heads" *Condition* and the *Condition Scenario* on the other branch will contain the "Tails" *Condition*. Only the *Effect Scenario* containing the "Heads" *Effect* will be propagated along the branch with the "Heads" *Condition Scenario* and vice versa for the branch with the "Tails" *Condition Scenario*.

Definition 3.4.2.1. Condition Scenario
A state description containing only *Conditions* is referred to as a *Condition Scenario*. A *Condition Scenario* is a set of *Conditions* that represent a partial state description. A *Condition Scenario* dictates which *Effect Scenarios* are passed along a *sequence flow* to the next activity. Only *Effect Scenarios* that contain every *Condition* in a *Condition Scenario* qualify as valid descriptions of the state of the world under those conditions.
Let CS be a *Condition Scenario*. Let c_i be a *Condition* where i is an index in CS such that $CS = \{c_1, c_2, \ldots, c_n\}$
A *Condition Scenario* may also be referred to as a conjunction of *Conditions* $(c_1 \wedge c_2 \wedge \ldots \wedge c_n)$
All *Conditions* within a *Condition Scenario* must be consistent with a background

knowledge base KBR, i.e., no *Condition* can contradict directly or indirectly any other *Condition* within the same *Condition Scenario*.
$CS \cup KBR$ is consistent.
The difference between *Effect Scenarios* and *Condition Scenarios* is the same as the difference between *Effects* and *Conditions*. Syntactically they are identical but they differ in their semantics. A *Condition Scenario* is used to test an *Effect Scenario*. An *Effect Scenario* will only satisfy a *Condition Scenario* iff all *Conditions* in the *Condition Scenario* exist in the *Effect Scenario*. Let ES be a Effect Scenario. We say that ES satisfies CS iff $ES \vdash CS$
Condition Scenarios are annotated to the outgoing edges of decision gateways. Each outgoing edge will contain at most one *Condition Scenario*.

3.4.3 Immediate Effect Scenario

Before discussing an *Immediate Effect Scenario* an *Immediate Effect* must first be defined.

Definition 3.4.3.1. Immediate Effect
Let T_1 be an activity in a BPMN model. Let e be an *Effect*. We say that e is an *Immediate Effect* iff e is the direct consequence of T_1 and that e has been annotated to T_1. A BPMN activity can have zero to many *Immediate Effects* annotated to it.

Immediate Effects are the business analyst's access window into the worlds created by a business process model. A business analyst consults with an expert about an activity, performed by the expert, then annotates the *Immediate Effects*, described by the expert, to the activity in the BPMN model. These annotated *Immediate Effects* are grouped into an *Immediate Effect Scenario*. An *Immediate Effect Scenario* is a partial state description reflecting only things that have changed as a consequence of the activity in question. They do not describe any state that existed prior to the activity.

An *Immediate Effect Scenario* is a consistent set of *Immediate Effects* corresponding to an observed state description that is a direct consequence of an action. The word "observed" is used to emphasise that the state description is incomplete (partial) and describes only the things of interest to the observer. To qualify as an *Immediate Effect Scenario* all *Immediate Effects* within the set must have been added to the set manually and be associated directly with the task or event to which they have been annotated. This differentiates an *Immediate Effect Scenario* from a *Cumulative Effect Scenario* (see 3.4.4).

In many cases the consequences of an activity are non-deterministic, e.g. the activity "Flip a coin" described in 3.4.2 can have two *Immediate Effect Scenarios*, the coin lands with either heads or tails facing up. Each activity in a BPMN model

may have multiple disjoint *Immediate Effect Scenarios* annotated to it, each being a unique observed state description.

Each BPMN activity will contain zero to many *Immediate Effect Scenarios*. *Immediate Effect Scenarios* must comply with the same rules as all *Effect Scenarios*.

Definition 3.4.3.2. Immediate Effect Scenario
Let KBR be a background knowledge base. Let IES be an *Immediate Effect Scenario* such that IES is a set of *Immediate Effects*. Let $ie_i \in IES$ be an *Immediate Effect* such that $IES = \{e_1, e_2, \ldots, e_n\}$.
An *Immediate Effect Scenario* may also be referred to as a conjunction of *Immediate Effects* such that $IES = (ie_1 \wedge ie_2 \wedge \ldots \wedge ie_n)$.
An *Immediate Effect Scenario* must be consistent with KBR such that $IES \cup KBR$ is consistent.
Let T_1 be an activity in a BPMN model. Let AES be the set of alternate *Immediate Effect Scenarios* of T_1. Let $(IES, T_1)_j$ be an *Immediate Effect Scenario* such that all *Immediate Effects* in IES have been annotated to T_1 and j is an index in AES.
Alternate *Immediate Effect Scenarios* for the activity T_1 are represented as a set $\{(IES, T_1)_1, (IES, T_1)_2, \ldots, (IES, T_1)_n\}$.
A set of alternate *Immediate Effect Scenarios* is a disjunction of *Immediate Effect Scenarios* such that $AES = ((IES, T_1)_1 \vee (IES, T_1)_2 \vee \ldots \vee (IES, T_1)_n)$ and $\nexists j\prime$ such that $(IES, T_1)_j \cup (IES, T_1)_{j\prime} \cup KBR$ is consistent.
Although *Immediate Effect Scenarios* are disjoint an *Immediate Effect* can exist in more than one *Immediate Effect Scenario*. Let IE be a set of identical *Immediate Effects*. Let n be the cardinality of AES. Let m be the cardinality of IE. Let ie_j be an *Immediate Effect* where j is an index in AES such that $ie_j \in (IES, T_1)_j$. It can therefore be said that $ie_j \in IE$ iff $m \leq n$ and $\nexists j\prime(j = j\prime)$. It is quite common for *Immediate Effects* to occur in more than one *Immediate Effect Scenario*.
An *Immediate Effect Scenario* is not closed under logical consequences.

An *Immediate Effect Scenario*, being partial state description, contributes new information to an already existing state description. This existing state description is referred to as a *Cumulative Effect Scenario* and it is the result of an accumulation function.

3.4.4 Cumulative Effect Scenario

Before discussing a *Cumulative Effect Scenario* a *Cumulative Effect* must first be defined.

Definition 3.4.4.1. Cumulative Effect
Let KBR be a background knowledge base. Let T_1 be an activity in a BPMN model. Let PES be an *Effect Scenario* describing the current state of the world prior to T_1 being executed. Let IES be an *Immediate Effect Scenario* containing *Immediate Effects* annotated to T_1. Let CES be an *Effect Scenario* representing the consequent state of the world immediately following the execution of T_1. Let $acc(\ )$ be a belief revision function such that $acc(PES, IES) \vdash CES$ and $CES \cup KBR$ is consistent. Let e be an *Effect*. We say that e is a *Cumulative Effect* when $e \in CES$. We say e is a previous Cumulative Effect when $e \in PES$. A *Cumulative Effect* in CES may originate from PES, IES or KBR.

A *Cumulative Effect Scenario* is a set of *Effects* that is consistent with the background knowledge base and is a derived description of the state of the world as it exists after one or more activities have occurred. A *Cumulative Effect Scenario* can be an empty set $\{\emptyset\}$ in the case of the previous *Cumulative Effect Scenario* of a Start Event. In the majority of cases though a *Cumulative Effect Scenario* will be the result of a pair-wise accumulation function whose arguments are a *Cumulative Effect Scenario* from a previous activity and an *Immediate Effect Scenario* from the current activity being accumulated. We define "previous" in this statement to mean the activity immediately preceding the current activity in the sequence of execution denoted by the BPMN model.

Definition 3.4.4.2. Cumulative Effect Scenario
Let KBR be a background knowledge base. Let T_1 be an activity in a BPMN model. Let PES be an *Effect Scenario* describing the current state of the world prior to T_1 being executed. Let IES be an *Immediate Effect Scenario* containing *Immediate Effects* annotated to T_1. Let CES be an *Effect Scenario* representing the consequent state of the world immediately following the execution of T_1. Let $acc(\ )$ be a belief revision function such that $acc(PES, IES) \vdash CES$ and $CES \cup KBR$ is consistent. We say CES is a *Cumulative Effect Scenario* of T_1. The $acc(\ )$ function will generate one or more *Cumulative Effect Scenarios* for each activity in a BPMN model.
A *Cumulative Effect Scenario* is closed under logical consequences.

3.5 World List

We have stated that any activity in a BPMN model may be the cause of multiple alternate outcomes. The activity may postulate direct outcomes in the case of *Immediate Effect Scenarios* or it may generate outcomes in the case of *Cumulative Effect Scenarios*. Each outcome is a possible world [54] and each possible world is disjoint from every other possible world associated with a particular activity.

Definition 3.5.0.1. World List
Let WL_t be a set of *Effect Scenarios* of a fixed type t such that WL consists of all *Immediate Effect Scenarios* or all *Cumulative Effect Scenarios*.
Let ES_x be an *Effect Scenario* where x is a type, the value of which is either a *Cumulative Effect Scenario* or an *Immediate Effect Scenario*.
Let $ES_x \in WL_t$ iff $x = t$.
Let T_j be an activity in a BPMN model where j is an identifier of T.
Let $ES_{x,j}$ denote that ES_x is associated with T_j. It may be the case that $ES_{x,j}$ is an *Immediate Effect Scenario* or a *Cumulative Effect Scenario* of T_j
A *World List* is a disjoint set of *Effect Scenarios* associated with a single BPMN activity such that $WL_t = \{(ES_{x,j})_1 \vee (ES_{x,j})_2 \vee \ldots \vee (ES_{x,j})_n\}$
We say that WL_t qualifies as a *World List* iff $\forall i((ES_{x,j})_i \in WL_t) \mid x = t$ and j and t are constants.
Each activity in a BPMN model will be associated with at most one *World List* of *Immediate Effect Scenarios* and exactly one *World List* of *Cumulative Effect Scenarios*.

3.6 Accumulation

The Oxford Dictionary defines a Language (in relation to Computing) as:

> *A system of symbols and rules for writing programs or algorithms.*

The Merriam-Webster Dictionary defines Language (in relation to Computing) as:

> *A formal system of signs and symbols including rules for the formation and transformation of admissible expressions.*

The accumulation of *Effect Scenarios* involves a background knowledge base (KB) with which all *Effect Scenarios* must be consistent. The background knowledge base contains all terms used within the scope of a semantically annotated business process model repository. It also contains rules that govern the syntax of admissible expressions. However, it does not contain rules that govern the semantics or the transformation of admissible expressions. These rules are acquired from a separate

set of rules (R) and a repository of defined artifacts (A). Together they satisfy the definition of a Language $\mathcal{L}$. The set of rules (R) contains global rules that apply to all *Effect Scenarios*. In contrast the rules contained in the artifact repository are instance-based rules that apply only to specific state descriptions. Henceforth they will be referred to as *Artifact Rules*. A simple example of an artifact rule may be that if the current state of an artifact is s then if an action A is performed within an *Effect Scenario* containing s then s MUST be transformed into $s\prime$. The global rules repository (R) contains things like government legislation whereas an artifact repository will contain things like operational rules or physical laws that govern the transformation of an artifact. It is expected that conflicts will arise between these two sets of rules. Conflicts serve to improve compliance thus when a new piece of legislation is introduced, conflicts in the established repositories are highlighted. If the legislation conflicts with a physical law then the legislation cannot be enacted. If any existing rule, other than a physical law, is in conflict with a new piece of legislation then the existing rule must change.

Effect Scenarios are transformed by an accumulation process, part of which is a pair-wise accumulation function. The accumulation process is a complex set of algorithms for processing the different types of elements and structures found in BPMN. Accumulation can be performed using two methods. The first method computes what we refer to as *Scenario Labels* that direct the pair-wise accumulation process. The second method simply walks through the process performing pairwise accumulation at each progression.

3.6.1 Accumulating with Scenario Labels

A *Scenario Label* is a precise, partially ordered, set (poset) of tasks that define a path leading from the *Start Event* in a model to the selected task. The simplest form of *Scenario Label* is a sequence of tasks. For example, $\langle S, T1, T2\rangle$ is a *Scenario Label* where S is the start event. A *Scenario Label* can either be a sequence, denoted by the $\langle\ \rangle$ delimiters, or a set denoted by the $\{\ \}$ delimiters or combinations of both. The set delimiters are used to deal with parallel splits, and distinct elements in a set can be performed in any order.

The ProcessSEER framework [63] is implemented as a tool that allows practitioners to annotate semantic effects to process activities/tasks and performs on-demand, anytime computation of *Cumulative Effects*. There are two stages to effect accumulation using *Scenario Labels*. The first stage in effect accumulation involves deriving a *Scenario Label*[52] which provides the organising locus for our procedure. Obtaining an *Effect Scenario* at a given point in a process involves a computation of the set of *Scenario Labels* at that point. Gateways introduce more complexity

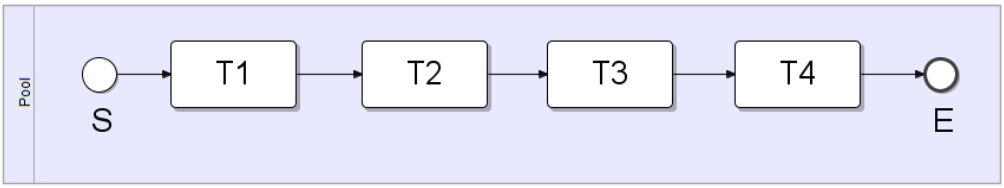

Figure 3.7: A BPMN model showing four activities in sequence.

into the computation of Scenario Labels.

Using Fig:3.7 as an example the second stage of effect accumulation involves the processing of *Immediate Effect* annotations for each of the tasks listed in the *Scenario Label* using a pair-wise operation where a *Cumulative Effect Scenario* of $T1$ is accumulated with an *Immediate Effect Scenario* of $T2$, the result being a *Cumulative Effect Scenario* at $T2$. The *Cumulative Effect Scenario* at $T2$ is then accumulated with an *Immediate Effect Scenario* of $T3$ resulting in a *Cumulative Effect Scenario* at $T3$ and so on up to T_n.

Contiguous Tasks: We define a process for *pair-wise effect accumulation*, which, given an ordered pair of tasks with effect annotations, determines the cumulative effect after both tasks have been executed in contiguous sequence. It is assumed throughout this document, the existence of a background knowledge-base (KB) that provides an additional basis for consistency. Consider the following simple example, where task $T2$ follows task $T1$, such that $T2$ somehow "undoes" the effects of $T1$ or changes the status of some entity referred to in $T1$. For instance, the status of a cheque submitted in $T1$ might be "not yet cleared", while the immediate effect of the "Clear cheque" task $T2$ might be to set its status to "cleared". A background rule that specifies that a cheque cannot have a "cleared" and "not yet cleared" status simultaneously ensures that we do not counter-intuitively obtain both status descriptions in the same *Effect Scenario*.

The procedure serves as a methodology for analysts to follow if only informal annotations are available. It is assumed that the effect annotations have been represented in conjunctive normal form (CNF) where each clause is also a *prime implicate* [120] (this provides a non-redundant canonical form). Simple techniques exist for translating arbitrary sentences into the conjunctive normal form, and for obtaining the prime implicates of a theory (references omitted for brevity).

Let $\langle T_i, T_j \rangle$ be an ordered pair of tasks connected via a sequence flow such that T_i precedes T_j, let e_i be an effect scenario associated with T_i and e_j be the immediate effect annotation associated with T_j. Let $e_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$ and $e_j = \{c_{j1}, c_{j2}, \ldots, c_{jn}\}$. If $e_i \cup e_j$ is consistent, then the resulting cumulative effect, denoted by $acc(e_i, e_j)$, is $e_i \cup e_j$. Else, we define $e'_i \subseteq e_i$ such that $e'_i \cup e_j$ is consistent and there exists no e''_i such that $e'_i \subset e''_i \subseteq e_i$ and $e''_i \cup e_j$ is consistent. We define

$acc(e_i, e_j) = e_i' \cup e_j$. Note that $acc(e_i, e_j)$ is non-unique i.e. there are multiple alternative sets that satisfy the requirements for e_i. In other words, the cumulative effect of the two tasks consists of the effects of the second task plus as many of the effects of the first task as can be consistently included. The procedure removes those clauses in the effect annotation of the first task that contradict the effects of the second task. The remaining clauses are undone, i.e., these effects are overridden by the second task.

In the preceding, it is assumed that all consistency checks implicitly include a background knowledge base (KBR) containing rules and axioms. Thus, the statement that $e_i' \cup e_j$ is consistent, effectively entails $e_i' \cup e_j \cup KBR$ is consistent. The following example illustrates an application of this definition.

Example:

Let e_1 and e_2 represent *Effect* annotations at $T1$ and $T2$ in a process model where $T2$ immediately follows $T1$. Let e_1 represent a *Cumulative Effect Scenario* while e_2 represents an *Immediate Effect Scenario*. At $T1$ the *Cumulative Effect Scenario* is $(p \wedge q)$ and the *Immediate Effect Scenario* of $T2$ is r. A rule exists in the KBR that states $KBR = r \rightarrow \neg(p \wedge q)$.

$e_1 = (p \wedge q)$

$e_2 = r$

$KBR = r \rightarrow \neg(p \wedge q)$

$\neg(p \wedge q) \equiv (\neg p \vee \neg q)$

Applying the definition above, the two alternative *Effect Scenarios* describing the *Cumulative Effects* at $T2$ are $\{p, r\}$ and $\{q, r\}$.

Much of the earlier and following discussion pertains to flows within individual pools. Message flow links across pools can be dealt with in a relatively straightforward fashion by requiring an *Immediate Effect* annotation for each incoming message. These *Effects* are combined via conjunction with the *Immediate Effects* of the task associated with the incoming message. Once again it is assumed that no inconsistencies appear between the message and task effects – such inconsistencies would only appear in erroneous process designs.

The procedure described above does not satisfactorily deal with loops, but can perform approximate checking by partial loop unravelling. Some *Effect Scenarios* generated using this approach might be infeasible. Note that the objective is to devise decision-support functionality in the compliance management space, with human analysts vetting key changes before deployment.

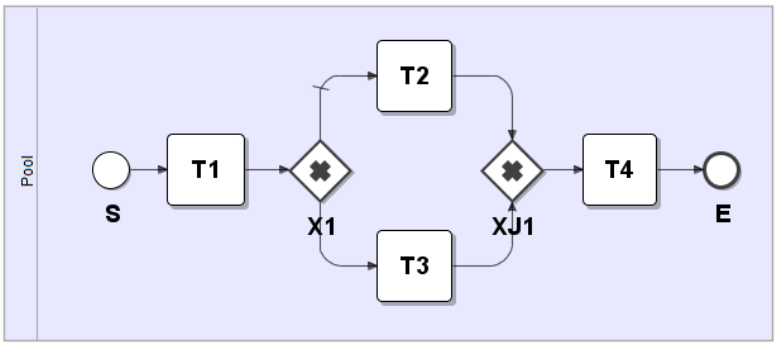

Figure 3.8: A simple exclusive *Gateway* structure with only two alternative *Activities*.

3.6.2 Accumulation Functions

Accumulation is performed across an entire BPMN model but for the purpose of explaining the functions involved a single instance of accumulation is used, i.e., when the *Immediate World List* of a single activity is accumulated with the previous *Cumulative World List*. An accumulation instance involves three functions, a pair-wise accumulation function *acc*( ), a decision making function *dec*( ) and a combinatorial function *combo*( ). For simplicity the descriptions of these functions restricts explanation to sets of *Effect Scenarios* rather than sets of *Ancestor Sequences* (see 3.7.2). In practice accumulation is performed on sets of *Ancestor Sequences*.

3.6.3 Decision Function

The first step in an accumulation instance involves determining the previous *Cumulative Effect Scenarios*. The first step is to determine the type of BPMN element immediately preceding the selected element. If the immediately preceding element is a decision gateway then the first accumulation function utilised is the *dec*( ) function (see Appendix A.1). Using Fig:3.8 as an example the activity label at $T1$ will indicate a decision to be made and $T1$ will be annotated with alternate *Immediate Effect Scenarios*, i.e., an *Immediate World List*. The accumulation at $T1$ will generate alternate *Cumulative Effect Scenarios*, i.e., a *Cumulative World List*. Using this approach gateways are never annotated, therefore an *Immediate Effect Scenario* of a gateway is an empty set. Having no *Immediate Effect Scenario*, associated with $X1$, to accumulate, the previous *Cumulative Effect Scenarios* from $T1$ are simply propagated as the current *Cumulative Effect Scenarios* of the gateway $X1$. That means that the current *Cumulative Effect Scenarios* at $X1$ are identical to the current *Cumulative Effect Scenarios* at $T1$.

Recall that each outgoing edge of a decision gateway has a single set of annotated *Conditions*, i.e., a *Condition Scenario*. The *Conditions* are used to filter the

Cumulative Effect Scenarios of a decision gateway. The *dec*( ) function takes two arguments, a previous *World List* from a decision gateway and a *Condition Scenario* from an outgoing branch of the decision gateway. The function returns a *World List* for the branch associated with the *Condition Scenario*. The *Condition Scenario* is a subset or equal to each *Cumulative Effect Scenario* in the returned *World List* indicating that a *Cumulative Effect Scenario* satisfies the *Condition Scenario*.

3.6.4 Combinatorial Function

The *combo*( ) function (see Appendix A.2) takes two arguments, the previous *World List* pWL (a set of *Cumulative Effect Scenarios*) and the *Immediate World List* iWL (a set of *Immediate Effect Scenarios*). The *combo*( ) function groups *Effect Scenarios* from each *World List* into pairs in a Cartesian product style matching between pWL and iWL. Each pair is a sequence $\langle pCES, IES \rangle$ containing a previous *Cumulative Effect Scenario* $pCES$ and an *Immediate Effect Scenario* IES. Each *Effect Scenario* pair is passed to the *acc*( ) function (see Appendix A.4) which returns a *World List*. The *Cumulative Effect Scenarios* in each *World List* derived from a pair are added to the final current *World List* cWL of the current activity. Each *Cumulative Effect Scenario* in the current *World List* is an alternate state description representing one possible outcome had the process executed up to and including the current activity.

3.6.5 Pair-wise Accumulation Function

The *acc*( ) function (see Appendix A.4) performs pair-wise accumulation on two *Effect Scenarios*, a previous *Cumulative Effect Scenario* and an *Immediate Effect Scenario*. The pair-wise accumulation of *Effect Scenarios* involves more than just asserting the *Effects* from the *Immediate Effect Scenario* into the previous *Cumulative Effect Scenario*. For example, consider the pair-wise accumulation function $cWL = acc(pCES, IES)$ where cWL is the current *World List* of $cCES$ (current *Cumulative Effect Scenarios*), $pCES$ is the previous *Cumulative Effect Scenario* and IES the *Immediate Effect Scenario*. Now apply the following value assignments:
$pCES = p$
$IES = \neg p$
Monotonic reasoning would result in $cCES = p \wedge \neg p$ which is inconsistent whereas non-monotonic reasoning would give us the correct result of $cCES = \neg p$. All *Effects* in IES must hold in $cCES$. Pair-wise accumulation is not as simple as replacing one *Effect* with another. For example, take two sequential activities, the first $T1$ is labelled "Paint object red" and the second $T2$ is labelled "Paint object green". A *Cumulative Effect Scenario* of $T2$, after $T1$ has been accumulated, could be either of the following:

1. $\{(object, hasColour, red), (object, hasColour, green)\}$ or
2. $\{(object, hasColour, green)\}$ or
3. $\{(object, hasColour, brown)\}$.

In the first case a rule stipulates that the object is allowed to have more than one colour. In the second case, painting the object replaces any existing colour and in the third case, the colours are mixed. The *acc*( ) function is dependent on the background knowledge base, rule repository and artifact repository KBR for delivering the correct *Cumulative Effect Scenario*. If the union of $pCES$ and IES is inconsistent then the *acc*( ) function passes the inconsistent *Effect Scenario* to a *Possible Worlds* function (see Appendix A.3) that returns a set of *Cumulative Effect Scenarios* that are consistent maximal subsets of the inconsistent *Effect Scenario*. The *Possible Worlds* function minimises the number of consistency checks required by first testing whether each new permutation of the inconsistent *Effect Scenario* is a subset of already established subsets in the return set. If a maximal subset, that includes the permutation, already exists then no consistency check is required and the permutation is discarded. In most cases it was found that the $acc(pCES, IES)$ returns a single *Cumulative Effect Scenario* from the *Possible Worlds* function. However, the function can return more than one consistent maximal subset thus expanding the number of *Cumulative Effect Scenarios* but possibly identifying previously unconsidered outcomes.

3.7 Gateway Structure

This document only covers instances of Exclusive, Inclusive and Parallel gateways. A *Gateway Structure* is marked by a gateway split at its beginning and a gateway join at its end (see Fig:3.9). Informal joins and gateway splits without a corresponding join are not covered in this document. A beginning gateway will split an incoming *World List* in different ways depending on the type of gateway. Different possible worlds will be propagated along the branches by exclusive and inclusive gateway splits while identical copies of the incoming *World List* will be propagated along the branches by a parallel gateway split. *Gateway Structures* may also be nested. It is necessary to think of a *Gateway Structure* like a subprocess (see Fig.3.9). A possible world enters a subprocess from one end, is changed within the subprocess and its changed state exits from the other end. The changed state may have split into multiple alternate possible worlds but they all originated from the one possible world. A similar thing occurs when accumulating a single previous *Cumulative Effect Scenario* with an activity that has two alternate *Immediate Effect Scenarios*.

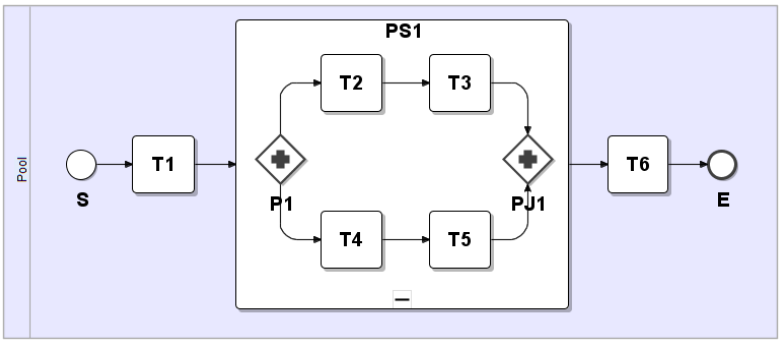

Figure 3.9: A Parallel Gateway Structure represented as a subprocess.

The previous *Cumulative Effect Scenario* is accumulated with each of the *Immediate Effect Scenarios* to produce two *Cumulative Effect Scenarios*. The activity example is the simple case while the subprocess example illustrates the complexity of what can happen as a single possible world is acted upon by many different activities, not all of them in sequence.

3.7.1 Accumulation over Gateway Structures

The base accumulation procedure becomes more complex when BPMN structures are introduced. A *Gateway Structure* is defined here as beginning with a *Gateway Split* and ending with a corresponding *Gateway Join* including all sequence flows and activities that occur in between. *Gateway Structures* can be nested. Algorithms are provided for the three most commonly used gateways in the BPMN standard, an *exclusive gateway*, an *inclusive gateway* and a *parallel gateway*. Although the BPMN syntax allows informal joins and splits, the methods described in this document only consider *Gateway Structures*. Accumulating over these structures requires further modifications to the underlying elements in the accumulation procedure.

A *Cumulative Effect Scenario* is a snapshot of the world that has resulted from an action occurring in a previous state of the world. When a *Cumulative Effect Scenario* passes through a parallel or inclusive gateway split it is literally split. A copy of the *Cumulative Effect Scenario* is propagated along each outgoing branch. It is important to understand that each of these copies is the same world acted upon by different activities. When each copy arrives at the corresponding gateway join, they need to be reassembled. The problem is the copies are individually altered by activities on different paths of the Gateway Structure so when they arrive at the corresponding gateway join and there are multiple copies on each branch, it becomes impossible to identify which copy should be reassembled with which.

Each *Cumulative Effect Scenario* needs to be tracked as it passes through a *Gateway Structure* so that it can be matched to its corresponding copy on other branches. This could be easily achieved by tagging *Cumulative Effect Scenarios* with

a serial number but a serial number does not contain information about what the Cumulative Effect Scenario was like before it was split. When tasks are performed in parallel, BPMN does not stipulate whether they occur concurrently so they may occur at exactly the same time or at different times. This freedom of execution means that the tasks in parallel *Gateway Structures* can be executed in any order or all at once. Every possible ordering requires a different mapped pathway through the structure. These mapped pathways were referred to in [63] as *Scenario Labels*. The problem with this method is that the number of *Scenario Labels* increases exponentially as the number of activities in the *Gateway Structure* increases.

With the understanding that *Effects* describe attributes or relationships of things in the real world, the possibility exists for the same attribute or relationship of a thing to be changed by different activities on different branches of a *Gateway Structure*. In a concurrent execution of parallel activities, this would amount to turning a light switch on and off at the same time which is impossible in the real world. Under non-concurrent execution, the order of execution will influence whether the light switch is on or off at the conclusion of the *Gateway Structure*. If a business analyst cannot determine the outcome of a business process, it makes no sense to design the process in the first place. Changes to the same attribute or relationship of the same thing on different branches of a *Gateway Structure* are therefore considered to be erroneous and should be brought to the attention of the business analyst designing the process.

Detecting changes to identical artifacts on different branches can only be achieved by comparing them to their original states, i.e., the state they were in prior to entering the *Gateway Structure*. This requires a record to be kept of every *Cumulative Effect Scenario*. When it comes time to reassemble Cumulative Effect Scenarios that have been split by a gateway, the records can be compared to determine whether changes have occurred to the same artifact on different branches. The records of changes to *Cumulative Effect Scenarios* are referred to as *Ancestor Sequences*.

3.7.2 Ancestor Sequences

Accumulation over parallel and inclusive *Gateway Structures* requires the ancestry of an *Effect Scenario* to be tracked throughout a BPMN model. Each time an *Effect Scenario* is accumulated its revised state is added to an *Ancestor Sequence*. An *Ancestor Sequence* represents an evolutionary timeline of one particular *Effect Scenario*. The last *Effect Scenario* in the sequence represents the current state and the first *Effect Scenario*, its origin. All *Effect Scenarios* in an *Ancestor Sequence* other than the last *Effect Scenario* are referred to as an *Effect Scenario History* of the last *Effect Scenario* in the *Ancestor Sequence*. *Ancestor Sequences* contain only

Cumulative Effect Scenarios. *Immediate Effect Scenarios* do not occur in *Ancestor Sequences*.

Definition 3.7.2.1. Ancestor Sequence
An *Ancestor Sequence* is a sequence of *Effect Scenarios*. Every *Effect Scenario* in an *Ancestor Sequence* is the same *Effect Scenario* as it existed at different sequential points in a process. An *Ancestor Sequence* is a record of the instances of a single *Effect Scenario* as it changes after each accumulation.
Let es be an *Effect Scenario*.
Let as be an *Ancestor Sequence* such that $as = \langle es_1, es_2, \ldots, es_n \rangle$ where index 1 is the oldest ancestor of an *Effect Scenario* and index n is the most current.
We say $\langle es_1, es_2, \ldots, es_{n-1} \rangle$ is the *Effect Scenario History* of es_n

With the introduction of *Ancestor Sequences* a *Cumulative World List* becomes a set of *Ancestor Sequences* whereas an *Immediate World List* remains a set of *Immediate Effect Scenarios*. This alters the algorithms used for accumulation. For example, we need to extend the *acc*( ) function to accept previous *Ancestor Sequences* instead of previous *Cumulative Effect Scenarios* (see Appendix A.5). One more element needs to be introduced into the mix. *Effect Scenarios* that are split at the beginning of a parallel or inclusive *Gateway Structure* must be reassembled at the corresponding join. Now that it is possible to detect the origin of a *Cumulative Effect Scenario* by its *Ancestor Sequence*, matching *Cumulative Effect Scenarios* are collected into what is referred to as a *Branch Combination*.

3.7.3 Branch Combinations

Exclusive gateway splits do not split the workflow. The workflow will only proceed along one outgoing branch of an exclusive gateway split. A parallel gateway split will distribute workflow along every outgoing branch and an inclusive gateway split will distribute workflow along a variable number of outgoing branches depending on whether an *Effect Scenario* exists to satisfy the *Conditions* of the outgoing branch. When workflow is split an *Effect Scenario* is split into copies of itself. When the workflow encounters a corresponding gateway join, those split *Effect Scenarios* need to be reassembled. *Branch Combinations* are used to collect the split *Effect Scenarios* for reassembly.

Effect Scenarios change as they are accumulated along different branches so it's impossible to identify which *Effect Scenarios* should be included in a *Branch Combination*. The importance of *Ancestor Sequences* becomes evident when trying to match accumulated *Effect Scenarios* from different branches. The *Effect Scenario History* is used to identify *Ancestor Sequences* that have originated from the same

ancestor. Only *Ancestor Sequences* with identical *Effect Scenario Histories* qualify for reassembly.

A *Branch Combination* is a set of *Ancestor Sequences* containing exactly one *Ancestor Sequence* from each branch entering a parallel or inclusive gateway join. All *Ancestor Sequences* in a *Branch Combination* share the same *Effect Scenario History* (see 3.7.2.1). All *Branch Combinations* of a parallel gateway join (a *Parallel Branch Combination*) will contain exactly the same number of *Ancestor Sequences* as there are incoming branches (one from each branch) whereas *Branch Combinations* of an inclusive gateway join (an *Inclusive Branch Combination*) can contain varying numbers of *Ancestor Sequences* depending on how many branches contain *Ancestor Sequences* with *Effect Scenarios* that satisfy the *Conditions* of the branch. An Ancestor Sequence will not be split evenly across all branches of an inclusive *Gateway Structure*.

Definition 3.7.3.1. Parallel Branch Combination
A *Parallel Branch Combination* is a set of *Ancestor Sequences* containing exactly one *Ancestor Sequence* from each incoming *branch* and each *Ancestor Sequence* shares the same *Effect Scenario History* (see 3.7.2.1).
Let B be a set of branches entering a parallel or inclusive gateway join.
Let b_x be a branch such that $b_x \in B$ and x is an index.
Let (as_x) be an *Ancestor Sequence* where x denotes an association with b_x.
Let (es_x) be an *Effect Scenario* such that $(es_x) \in (as_x)$ where x denotes an association with (as_x).
Let t be an integer denoting the last element in (as_x) such that
$(as_x) = \langle (es_x)_1, (es_x)_2, \ldots, (es_x)_t \rangle$ and $t = |(as_x)|$.
Let h be an *Effect Scenario History* such that h is a sequence $\langle (es_x)_1, (es_x)_2, \ldots, (es_x)_{t-1} \rangle$.
Let PBC_h be a *Parallel Branch Combination* such that $|PBC| = |B|$ and where h denotes a qualifying sequence.
Let i be an index in PBC_h
$(as_x)_i \in PBC_h$ iff $\forall i(h \subset (as_x)_i)$ and $i = x$.

Definition 3.7.3.2. Inclusive Branch Combination
An *Inclusive Branch Combination* is a set of *Ancestor Sequences* containing at most one *Ancestor Sequence* from each incoming branch and each *Ancestor Sequence* shares the same *Effect Scenario History* (see 3.7.2.1).
Let B be a set of branches entering a parallel or inclusive gateway join.
Let b_x be a branch such that $b_x \in B$ and x is an index.
Let (as_x) be an *Ancestor Sequence* where x denotes an association with b_x.
Let (es_x) be an *Effect Scenario* such that $(es_x) \in (as_x)$ where x denotes an association with (as_x).

Let t be an integer denoting the last element in (as_x) such that
$(as_x) = \langle (es_x)_1, (es_x)_2, \ldots, (es_x)_t \rangle$ and $t = |(as_x)|$.
Let h be an *Effect Scenario History* such that h is a sequence $\langle (es_x)_1, (es_x)_2, \ldots, (es_x)_{t-1} \rangle$.
Let IBC_h be an *Inclusive Branch Combination* such that $|IBC| \leq |B|$ and where h denotes a qualifying sequence.
Let i be an index in IBC_h
$(as_x)_i \in IBC_h$ iff $\forall i(h \subset (as_x)_i)$ and $i = x$.

3.8 Accumulating BPMN Elements

This section looks at different sequences of two BPMN elements and how that affects accumulation algorithms. Accumulation in its most basic form is the transformation of one world into another based on a reasoning process. We know that actions in the world can have different outcomes so when propagating these outcomes across a business process model, an activity may be presented with multiple worlds in which it could be executed. Accumulation suddenly becomes an operation of transforming multiple worlds into multiple worlds instead of one world into multiple worlds. In real life we do something and observe the outcome then based on that outcome we choose to do something else so in real life there is constant monitoring and updating. At each check point there is only one *Effect Scenario* to consider, the one that is factual, not the others that are theoretical. This describes the difference between run-time BPMN models and design-time BPMN models. At design-time we are concerned about all the possible outcomes that could occur and how we plan to respond to them. The following notation applies to the different BPMN elements:

T A task or event (an activity)

X An exclusive gateway split

XJ An exclusive gateway join

I An inclusive gateway split

IJ An inclusive gateway join

P A parallel gateway split

PJ A parallel gateway join

B A set of incoming branches to a gateway join

Pair-wise accumulation of two BPMN elements in sequence is denoted by $\langle E_1, E_2 \rangle$ where E is a BPMN element, e.g., $\langle T, T \rangle$ indicates that an accumulation is being

performed on the second activity using the *Cumulative Effect Scenarios* from the first activity and the *Immediate Effect Scenarios* from the second activity. Some algorithms apply to more than one of these BPMN element sequences. The sequences to which the algorithm applies are listed in each section.

3.8.1 Accumulating an Activity following another Activity

Let's start with the simplest of accumulation procedures, two activities $T1$ and $T2$ in sequence. In this case we are referring to tasks but these could easily be replaced by events or any combination of the two. It has been established that the *Cumulative World List* that was accumulated from $T1$ is the previous *Cumulative World List* of $T2$. *Cumulative Effect Scenarios* in the previous *Cumulative World List* are accumulated with the *Immediate Effect Scenarios* in the *Immediate World List* from $T2$ to produce a current *Cumulative World List* for $T2$. The algorithm (see Appendix A.6) applies to the following accumulation instances: $\langle T, T\rangle$, $\langle XJ, T\rangle$, $\langle PJ, T\rangle$, $\langle IJ, T\rangle$.

3.8.2 Accumulating a Parallel or Inclusive Gateway Split

Accumulating *Effect Scenarios* for a parallel gateway split is comparatively easy compared to other types of accumulation. The parallel gateway split contains no *Immediate World List* but at this point it is necessary to start tracking a *Cumulative Effect Scenario*. For simplicity we will assume that all *Ancestor Sequences* in the previous *Cumulative World List* contain only a single *Cumulative Effect Scenario*. The *Cumulative Effect Scenario* is copied and added to the *Ancestor Sequence*. Doing this allows the last *Cumulative Effect Scenario* (the copy recently added) in the *Ancestor Sequence* to be accumulated while retaining its original form as the first element in the sequence. The algorithm (see Appendix A.7) applies to the following accumulation instances: $\langle T, P\rangle$, $\langle XJ, P\rangle$, $\langle PJ, P\rangle$, $\langle IJ, P\rangle$, $\langle T, I\rangle$, $\langle XJ, I\rangle$, $\langle PJ, I\rangle$, $\langle IJ, I\rangle$, $\langle P, I\rangle$.

3.8.3 Accumulating an Exclusive Gateway Split

There is no need to track a *Cumulative Effect Scenario* through an exclusive *Gateway Structure* because the *Cumulative Effect Scenario* is never split. It can only be propagated along a single branch so it never has a counter-part to be reassembled with. This makes accumulation at an exclusive gateway split the simplest of all types of accumulation. The previous *Cumulative World List* is simply copied to become the current *Cumulative World List*. The algorithm (see Appendix A.8) applies to the following accumulation instances: $\langle T, X\rangle$, $\langle XJ, X\rangle$, $\langle PJ, X\rangle$, $\langle IJ, X\rangle$, $\langle P, X\rangle$.

3.8.4 Accumulating an Activity following an Exclusive/Inclusive Gateway Split

The first activity that follows an exclusive gateway split must determine which of the previous *Cumulative Effect Scenarios* satisfy the *Conditions* of the branch. This generates a unique set of previous *Cumulative Effect Scenarios* for the incoming sequence flow to the activity. Accumulation can then proceed using the filtered set of previous *Cumulative Effect Scenarios*. The algorithm (see Appendix A.9) applies to the following accumulation instances: $\langle X, T \rangle$, $\langle I, T \rangle$.

3.8.5 Accumulating an Exclusive Gateway Split following an Exclusive/Inclusive Gateway Split

Gateway splits have no *Immediate World List* so the previous *Cumulative World List* is simply copied as the current *Cumulative World List*. When the preceding BPMN element is an exclusive gateway split or an inclusive gateway split, there are *Conditions* annotated to the sequence flow connecting the two gateway splits. A new previous *Cumulative World List* needs to be constructed that satisfies the *Condition Scenario* before it can be copied into the current *Cumulative World List* of the second gateway split. The algorithm (see Appendix A.10) applies to the following accumulation instances: $\langle X, X \rangle$, $\langle I, X \rangle$.

3.8.6 Accumulating a Parallel/Inclusive Gateway Split following an Exclusive/Inclusive Gateway Split

Gateway splits have no *Immediate World List* so the previous *Cumulative World List* is simply copied as the current *Cumulative World List*. When the preceding BPMN element is an exclusive gateway split or an inclusive gateway split, there are *Conditions* annotated to the sequence flow connecting the two gateway splits. A new previous *Cumulative World List* needs to be constructed that satisfies the *Condition Scenario* before it can be copied into the current *Cumulative World List* of the second gateway split. When the second gateway split is a parallel or inclusive gateway split then the last *Effect Scenario* in each *Ancestor Sequence* must be duplicated and added to the end of the *Ancestor Sequence* before copying it into the current *Cumulative World List*. The algorithm (see Appendix A.11) applies to the following accumulation instances: $\langle X, P \rangle$, $\langle I, P \rangle$, $\langle X, I \rangle$.

3.8.7 Accumulating an Inclusive Gateway Split

An inclusive *Gateway Structure* caters to *Cumulative Effect Scenarios* that satisfy more than one set of *Conditions*. A *Cumulative Effect Scenario* may split over two branches out of three branches or maybe only one branch out of three or perhaps all branches. The possibility of it splitting is why *Ancestor Sequences* are needed. For this reason, *Branch Combinations* at an inclusive gateway join do not have to contain an *Ancestor Sequence* from every branch. When an inclusive gateway split is the current BPMN element being accumulated with a preceding activity, the algorithm (see Appendix A.7) is used. When an inclusive gateway split is the preceding BPMN element being accumulated with a current activity, the algorithm (see Appendix A.9) is used.

3.8.8 Accumulating an Exclusive Gateway Join

Gateway joins have no *Immediate World List* so accumulation is a matter of combining the *Cumulative World Lists* entering the gateway join. Each incoming branch to an exclusive gateway join has an associated *Cumulative World List*. The *Ancestor Sequences* in each *Cumulative World List* are completely independent of any other *Ancestor Sequence* in any other branch *Cumulative World List*. No *Ancestor Sequence* in any branch *Cumulative World List* needs to be combined with any other *Ancestor Sequence*. At the exclusive gateway join the *Ancestor Sequences* only need to be grouped into a single set. The algorithm (see Appendix A.12) applies to the following accumulation instances: $\langle B, XJ \rangle$.

3.8.9 Accumulating a Parallel or Inclusive Gateway Join

Within this subsection the term "gateway join" refers to either a parallel or inclusive gateway join but not an exclusive gateway join. In [81], Koliadis et. al propose that *Effect* accumulation occurs separately along each parallel branch and at the parallel *Gateway* join the *Cumulative Effects* of each incoming edge are unioned together. Gateway joins have no *Immediate World List* so accumulation is a matter of combining the *Cumulative World Lists* entering the gateway join. Gateway joins must identify which *Cumulative Effect Scenarios*, entering from each incoming branch, have originated from the same *Cumulative Effect Scenario* entering at the gateway split. This is achieved by comparing the *Effect Scenario Histories* of *Ancestor Sequences*. An *Ancestor Sequence* with the same *Effect Scenario History* on a different branch becomes a candidate for inclusion into a *Branch Combination*. Exactly one candidate *Ancestor Sequence* from each branch *Cumulative World List* is added to a *Branch Combination*. Each *Branch Combination* must include exactly one *Ancestor*

Sequence from each branch *Cumulative World List* when accumulating a parallel gateway join. When accumulating an inclusive gateway join then *Branch Combinations* will contain at most one *Ancestor Sequence* from each branch *Cumulative World List*.

Multiple *Ancestor Sequences*, originating from the same *Cumulative Effect Scenario*, may have been generated on each branch so each one must be grouped with each instance of an original from other branches. Nested *Gateway Structures* compound the number of *Ancestor Sequences* that need to be matched. The explanations of algorithm use given here adopt the perspective of an individual performing a task, in which case it is unknown what is happening on other parallel branches. The individual only knows about what they have done and how it has affected the state of the world. This approach greatly improves computational efficiency but does not result in a complete list of possible *Effect Scenarios* for activities occurring within a parallel or inclusive *Gateway Structure*. For a complete list of *Cumulative Effect Scenarios* the scenario label approach is recommended (see section 3.6.1). The same algorithms are used in the scenario label approach but the method adversely affects computational efficiency.

Once a complete set of *Branch Combinations* has been created, the accumulation procedure merges the *Ancestor Sequences* in each *Branch Combination* into a single *Ancestor Sequence*. The resulting *Ancestor Sequence* is a union of all *Ancestor Sequences* in the *Branch Combination*. The union of *Ancestor Sequences* involves accumulation and replication. The last *Effect Scenario* of each *Ancestor Sequence* in a *Branch Combination* is unioned together then the *Effects* in the second last *Effect Scenario* in the *Ancestor Sequence* are removed from the union. The *Effects* of the second last *Effect Scenario* of all *Ancestor Sequences* in a *Branch Combination* should be identical because they were matched based on identical *Effect Scenario Histories*.

The second last *Effect Scenario* in an *Ancestor Sequence* represents the state of the *Effect Scenario* before it entered the parallel gateway split. It is only natural that some of these original *Effects* may have remained unchanged by activities within the *Gateway Structure*. If original *Effects* have been changed on a single branch in the parallel *Gateway Structure* then they will conflict with their unchanged versions on other branches where they have simply been propagated along the branch. This is why the original *Effects* need to be removed from the unioned *Effect Scenarios*. With no original *Effects* in the union, it is then tested for consistency. If it is inconsistent then an *Effect* attribute or relationship has been altered on different branches which indicates an erroneous design. The business analyst must correct the design to maintain control over the outcome.

If the union is consistent then the union must be accumulated with the original *Effect Scenario*. In this case the union acts like an *Immediate Effect Scenario* of the gateway join and the second last *Effect Scenario* in each of the *Ancestor Sequences* becomes the previous *Cumulative Effect Scenario* in the accumulation procedure. The resulting *Cumulative Effect Scenario* then replaces the last two *Effect Scenarios* in the *Ancestor Sequence*. The *Ancestor Sequence* is consequently reduced in size at the gateway join as are the number of *Ancestor Sequences* reduced when they are unioned together. This method treats the entire *Gateway Structure* as a single activity that is accumulated with whatever preceded it. The only difference between a parallel and an inclusive gateway join accumulation is that Branch Combinations must contain an Ancestor Sequence from each incoming branch whereas an inclusive gateway join will accept at most one Ancestor Sequence from each incoming branch.

The gateway join accumulation involves four steps:

1. Clustering – produces *Branch Clusters* (or just Clusters)
2. Grouping – produces *Branch Groups* (or just Groups)
3. Combining – produces *Branch Combinations*
4. Accumulating – produces *Cumulative Effect Scenarios*

Four additional functions are required to accumulate a parallel or inclusive gateway join. The first function clusters together *Ancestor Sequences* on the same branch that have originated from the same *Cumulative Effect Scenario* at the beginning of the *Gateway Structure*. This creates a set of *Clusters* for the incoming branch. The next function groups *Clusters* from each branch whose *Ancestor Sequences* originated from the same *Cumulative Effect Scenario* at the beginning of the *Gateway Structure*. This creates a set of *Branch Groups*. A *Branch Group* for a parallel gateway join will contain exactly one *Cluster* from each branch whereas *Branch Groups* for an inclusive gateway join will contain at most one *Cluster* from each incoming branch. The third function combines the *Ancestor Sequences* in each *Cluster* into all possible permutations. This creates *Branch Combinations* that contain exactly one *Ancestor Sequence* from each *Cluster* in the *Branch Group*. The final function then accumulates these *Branch Combinations* as described previously. The algorithm (see Appendix A.13) applies to the following accumulation instances: $\langle B, PJ \rangle$ and $\langle B, IJ \rangle$.

Diagram Fig:3.10 shows how the *Ancestor Sequences* are reassembled through the different stages of parallel join accumulation. The letters, A, B and C represent three original *Ancestor Sequences* that entered the parallel gateway structure. Note that on different branches some originals have become two alternate *Ancestor Sequences*, e.g., $A1$ and $A2$ on the top branch have both originated from an original

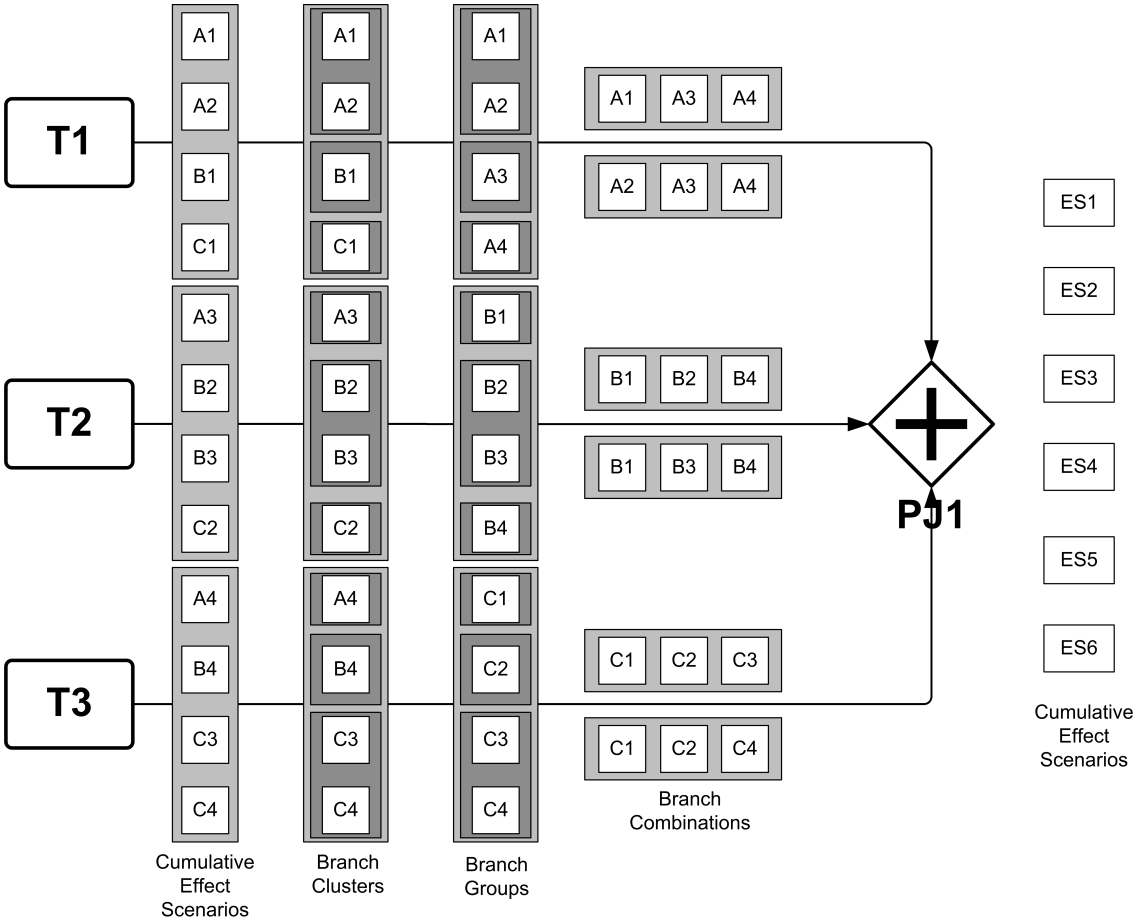

Figure 3.10: Diagram showing the breakdown of accumulating a BPMN parallel join.

Ancestor Sequence A. Some activity along that branch has generated two possible outcomes from the original state. The box surrounding the *Ancestor Sequences* represents a World List for each branch.

The diagram shows how clustering assembles the *Ancestor Sequences* for each branch. At this point the surrounding box of the *Branch Clusters* is still related to the branch. The boxes surrounding the *Branch Groups* is not related to the branch. It simply defines a *Branch Group*. *Branch Groups* contain *Clusters* from each branch. The boxes surrounding *Branch Combinations* also do not relate to the branches. They are simply containers for the *Branch Combinations*. The *Branch Combinations* show how the *Clusters* in each *Branch Group* affect the number of *Branch Combinations*.

Diagram Fig:3.11 shows how an inclusive gateway is accumulated. Note that certain *Ancestor Sequences* only appear on one or two branches due to the outgoing *Conditions* at the inclusive gateway split. Not all branches will contain matching *Ancestor Sequences* therefore *Branch Groups* can contain less *Ancestor Sequences* than the number of branches. The other steps mirror those of the parallel accumulation in Fig:3.10.

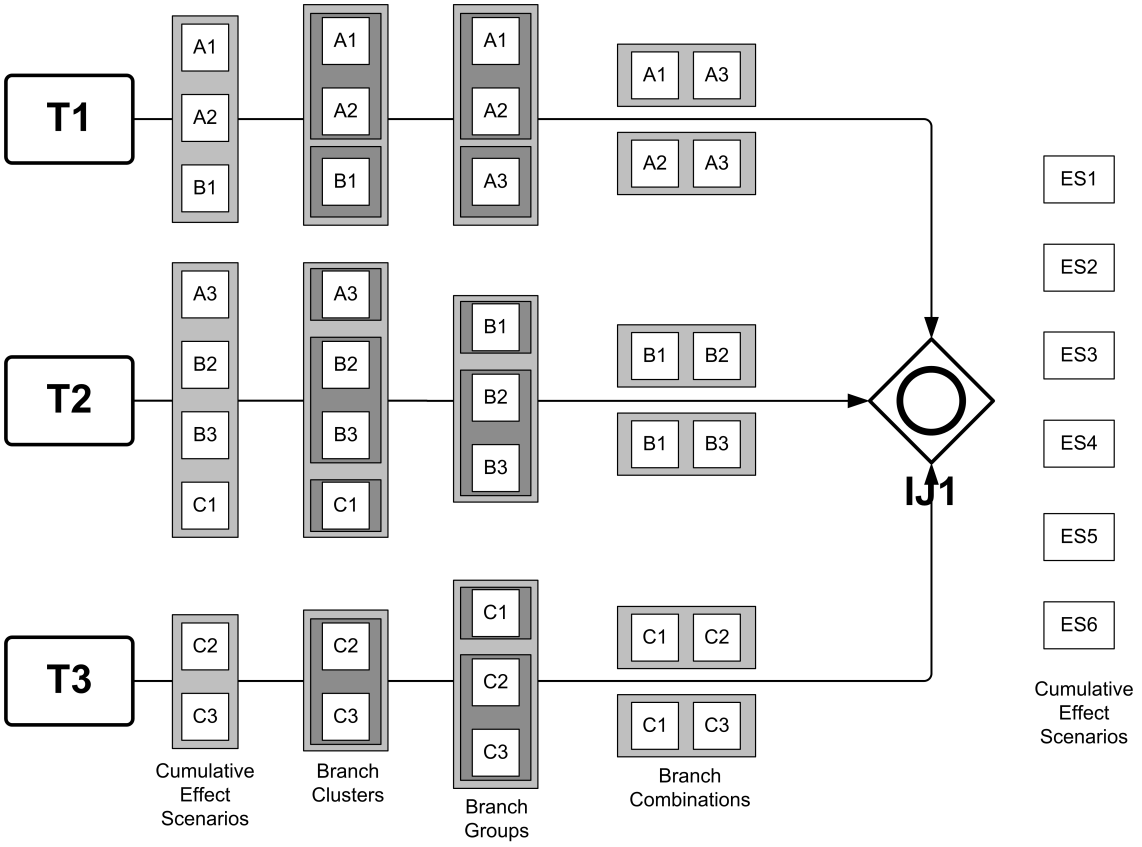

Figure 3.11: Diagram showing the breakdown of accumulating a BPMN inclusive join.

3.8.10 Branch Cluster

When alternate *Cumulative Effect Scenarios* are generated on a single branch they have a common origin. There may be multiple *Ancestor Sequences* entering a branch and each one could generate alternate *Ancestor Sequences*. The potential exists for multiple *Ancestor Sequences* that originated from the same *Ancestor Sequence* to arrive at a gateway join on a single branch. *Ancestor Sequences* from the same origin need to be grouped into what are referred to as *Clusters*. This naming practice has been used in an attempt to minimise confusion when describing sets within sets within sets so a *Cluster* is a set of *Ancestor Sequences* that all have the same *Effect Scenario History* and exist within a single *Cumulative World List* on an incoming branch to a gateway join.

Definition 3.8.10.1. Cluster

Let pWL be a previous *Cumulative World List* of one incoming branch to a gateway join.

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.

Let $pCES_n$ be the last *Cumulative Effect Scenario* in pAS.

Let pAS_h be an *Effect Scenario History* of pAS such that $pAS_h = pAS - pCES_n$.

Let H be an *Effect Scenario History* constant.

Let C be a set of *Ancestor Sequences*.
C is a *Cluster* iff $\forall pAS \in C(pAS_h = H)$.

3.8.11 Cluster Function

The *cluster*( ) function (see Appendix A.14) operates on a single branch *World List* and returns a set of *Clusters*. It first groups *Ancestor Sequences* with the same *Effect Scenario History* into a *Cluster* then assembles all the *Clusters* into a set. This transforms the previous *Cumulative World List* of an incoming branch from a set of *Ancestor Sequences* into a set of *Clusters*.

3.8.12 Branch Group

A *Branch Group* is a set of *Clusters*. A *Branch Group* for a parallel gateway join contains one *Cluster* from each incoming branch. A *Branch Group* for an inclusive gateway join contains at most one *Cluster* from each incoming branch. All *Ancestor Sequences* in all *Clusters* in a *Branch Group* share the same *Effect Scenario History*. They have originated from the same *Ancestor Sequence* that first entered the *Gateway Structure*.

Definition 3.8.12.1. Branch Group
Let B be a set of incoming branches to a gateway join.
Let $n = |B|$.
Let SC_i be a set of *Clusters* for a single incoming branch to a gateway join where i is an index.
Let C be a *Cluster* such that $C \in SC_i$.
Let cAS be an *Ancestor Sequence* such that $cAS \in C$.
Let $cCES_n$ be the last *Cumulative Effect Scenario* in cAS.
Let cAS_h be an *Effect Scenario History* of cAS such that $cAS_h = cAS - cCES_n$
Let H be an *Effect Scenario History* constant.
Let G be a set of Clusters.
G is a *Branch Group* iff $\forall C \in G(cAS_h = H)$ and $|G| = |B|$ and only one cluster from each branch.

3.8.13 Group Function

The *group*( ) function (see Appendix A.15) operates on all incoming branches to a gateway join. Before it can be utilised each *World List* on each incoming branch must be transformed into a set of *Clusters*. Each set of *Clusters* from each incoming branch is collected into a super set that is passed to the *group*( ) function as a

parameter. The function groups *Clusters*, from different branches, together. All *Ancestor Sequences* inside all *Clusters* inside a *Branch Group* must all have originated from the same *Cumulative Effect Scenario*. The *group*( ) function therefore collects one matching *Cluster* from each branch into a *Branch Group*. It then returns the complete set of all *Branch Groups*.

3.8.14 Join Accumulation Function

The *joinAcc*( ) function (see Appendix A.16) performs an accumulation on a single *Branch Combination*. A *Branch Combination* is a set of *Ancestor Sequences*, one from each incoming branch to a parallel or inclusive gateway join. All *Ancestor Sequences* in a *Branch Combination* have the same *Effect Scenario History*, i.e., they have originated from the same *Cumulative Effect Scenario* that entered the corresponding parallel or inclusive gateway split. The last *Cumulative Effect Scenario* in each of these *Ancestor Sequences* is the result of accumulation for the branch. If there has been no erroneously designed activities in the process model then all these last *Cumulative Effect Scenarios* can be consistently unioned together. If the union is inconsistent then the business analyst needs to be notified of the erroneous design.

The union of last *Cumulative Effect Scenarios* from each *Ancestor Sequence* will contain *Effects* that have remained unchanged throughout the accumulation along the branch. Some branches may have changed those *Effects* so when the last *Cumulative Effect Scenarios* are combined, the changed *Effects* from one branch will conflict with the unchanged *Effects* from another branch. It is therefore necessary to remove all the unchanged *Effects* from the union before checking it for consistency. The second last *Cumulative Effect Scenario* in each *Ancestor Sequence* is the parent of all the last *Cumulative Effect Scenarios* so the second last's *Effects* are removed from the union before checking for consistency.

The resulting union, containing no *Effects* from the second last *Cumulative Effect Scenario*, now becomes a substitute *Immediate Effect Scenario* for the gateway join. It contains only *Effects* that have been altered within the *Gateway Structure*. The next element to consider is the *Cumulative Effect Scenario* when it first entered the *Gateway Structure*, i.e., the second last *Cumulative Effect Scenario* in all the *Ancestor Sequences* in the *Branch Combination*. This becomes the previous *Cumulative Effect Scenario* that is accumulated with the *Immediate Effect Scenario* of the gateway join, i.e., the union. The result is a single *Cumulative Effect Scenario* for the gateway join or it could be thought of as a *Cumulative Effect Scenario* for the *Gateway Structure*.

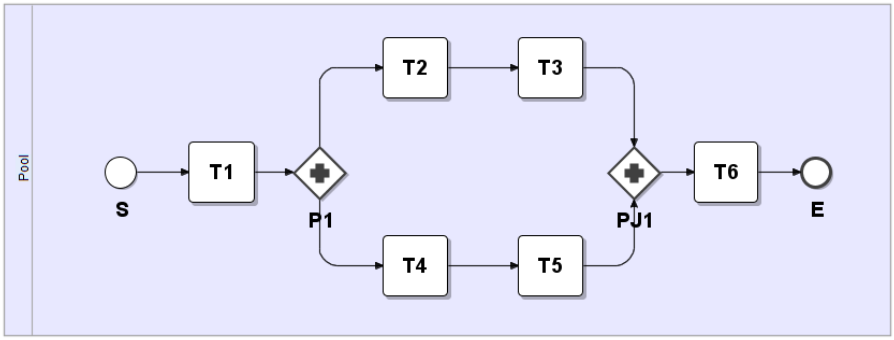

Figure 3.12: A parallel gateway structure containing sequential activities.

To maintain *Effect Scenario Histories* for nested *Gateway Structures* the last two *Cumulative Effect Scenarios* from any *Ancestor Sequence* in a *Branch Combination* (they will all be the same) are removed and the *Cumulative Effect Scenario* for the gateway join is added to the *Ancestor Sequence*. This then becomes one *Ancestor Sequence* in the *Cumulative World List* of the gateway join.

3.8.15 Branch Combinatorial Function

The *combo*( ) function (see Appendix A.2) needs to be modified so it can process *Clusters* inside *Branch Groups*. *Clusters* in *Branch Groups* can contain different numbers of *Ancestor Sequences*. The *combo*( ) function performs a matching of pairs similar to a Cartesian Product. Pair matching only involved two sets to combine. *Branch Groups* can contain multiple *Clusters* and all combinations of *Ancestor Sequences* between those *Clusters* must be matched into what is called a *Branch Combination*. Therefore the *branchCombo*( ) function (see Appendix A.17) must assemble more than just a pair of *Ancestor Sequences* into a *Branch Combination*. A *Branch Combination* will contain exactly one *Ancestor Sequence* from each *Cluster* in a *Branch Group* and each *Branch Combination* will be unique for the entire set of branches.

3.8.16 Parallelism with Scenario Labels

Scenario Labels containing parallel or inclusive *Gateway Structures* cannot be simply transformed into a set of sequences. As explained earlier, an *Effect Scenario* is split by a parallel or inclusive gateway. Each version of the same *Effect Scenario* is propagated along all branches of the *Gateway Structure*. Gateway joins reassemble these split *Effect Scenarios* which is not the same as accumulation. Therefore deriving only sequences from a *Scenario Label* containing parallel or inclusive *Gateway Structures* will result in only accumulation being performed at each step without the appropriate reassembly of split *Effect Scenarios*. Doing this will result in erroneous

outcomes. For example, the BPMN model in Fig:3.12 will generate the *Scenario Label*.
$\langle S, T1, P1, \{\langle T2, T3\rangle, \langle T4, T5\rangle\}, PJ1, T6, E\rangle$.
From this partially ordered set (poset), six sequences can be derived.

1. $\langle S, T1, P1, T2, T3, T4, T5, PJ1, T6, E\rangle$
2. $\langle S, T1, P1, T2, T4, T3, T5, PJ1, T6, E\rangle$
3. $\langle S, T1, P1, T2, T4, T5, T3, PJ1, T6, E\rangle$
4. $\langle S, T1, P1, T4, T5, T2, T3, PJ1, T6, E\rangle$
5. $\langle S, T1, P1, T4, T2, T3, T5, PJ1, T6, E\rangle$
6. $\langle S, T1, P1, T4, T2, T5, T3, PJ1, T6, E\rangle$

Each of these sequences represents a valid ordering of execution. Pair-wise accumulation at each step along the sequence prioritises the *Effects* of the current activity over the *Effects* of the previous activity. If, for example, the activity at $T2$ was to turn on a switch and the activity at $T4$ was to turn off the same switch then at $PJ1$ there can be two alternate *Effect Scenarios*, one where the switch is off and the other where the switch in on. Clearly the business analyst has no control over this outcome because different agents are allowed to affect the same artifact without any clearly defined ordering. Parallel *Gateway Structures* in BPMN do not indicate concurrency and even if they did it would be interesting to see what the outcome would be of two agents fighting over whether the switch should be on or off.

These sequences may offer a complete set of *Effect Scenarios* at each activity in a parallel *Gateway Structure* but they do not accurately represent the outcome at any given activity in the parallel *Gateway Structure*. The reason for this can be seen in the accumulation process of a parallel gateway join (see 3.8.9). Accumulation of *Effect Scenarios* from different branches of a parallel *Gateway Structure* involves a union of the *Effect Scenarios* from different branches accumulated with the original *Effect Scenario* as it entered the parallel *Gateway Structure*. This is because the *Effect Scenarios* on different branches are representative of the same world only with different things happening in them at the same time. They must eventually be reassembled before updating their original state. Deriving a set of sequences from a Scenario Label translates the actions of two separate agents into the actions of a single agent repeatedly executed in the different orders prescribed.

Knowing what the combined effects will be of two independently operating agents involves merging what is known about the world, by the two agents, into a single conjunction of *Effects* before attempting to update the world that was previously known by both agents. This procedure is not replicated in the *Scenario*

Label approach of separate sequences so parallel activities update each other. If there are no conflicting *Effects* from different branches then *Effects* from one branch are simply propagated during accumulation giving the same result as if the *Effect Scenarios* had been unioned. Checking for conflicting parallel *Effects* is much more complex using this *Scenario Label* method. Parallel activities must be identified so that joining procedures can be employed before accumulation procedures. Conflict checking occurs each time parallel *Effect Scenarios* are unioned so the number of paths increases and the number of conflict checks increases.

The original Scenario Label $\langle S, T1, P1, \{\langle T2, T3\rangle, \langle T4, T5\rangle\}, PJ1, T6, E\rangle$ provides a template for generating all possible accumulation methods for a single parallel activity. The following set of *Scenario Labels* are derived from the original and show the different execution paths for each parallel activity rather than the entire process. Parallel activities are denoted by { } and indicate a joining/union procedure whereas ⟨ ⟩ denotes sequential execution warranting accumulation procedures. Where two *Scenario Labels* are given for a single instance it means that the *Effect Scenarios* will be identical. These posets act as instruction sets controlling which accumulation functions are employed and in what order.

1. **T2** $\langle S, T1, P1, T2\rangle$
2. **T2** $\langle S, T1, P1, \{T2, T4\}\rangle$ or $\langle S, T1, P1, \{T4, T2\}\rangle$
3. **T2** $\langle S, T1, P1, \{T2, \langle T4, T5\rangle\}\rangle$ or $\langle S, T1, P1, \{\langle T4, T5\rangle, T2\}\rangle$
4. **T3** $\langle S, T1, P1, T2, T3\rangle$
5. **T3** $\langle S, T1, P1, \{\langle T2, T3\rangle, T4\}\rangle$ or $\langle S, T1, P1, \{T4, \langle T2, T3\rangle\}\rangle$
6. **T3** $\langle S, T1, P1, \{\langle T2, T3\rangle, \langle T4, T5\rangle\}\rangle$ or $\langle S, T1, P1, \{\langle T4, T5\rangle, \langle T2, T3\rangle\}\rangle$
7. **T4** $\langle S, T1, P1, T4\rangle$
8. **T4** $\langle S, T1, P1, \{T4, T2\}\rangle$ or $\langle S, T1, P1, \{T2, T4\}\rangle$
9. **T4** $\langle S, T1, P1, \{T4, \langle T2, T3\rangle\}\rangle$ or $\langle S, T1, P1, \{\langle T2, T3\rangle, T4\}\rangle$
10. **T5** $\langle S, T1, P1, T4, T5\rangle$
11. **T5** $\langle S, T1, P1, \{\langle T4, T5\rangle, T2\}\rangle$ or $\langle S, T1, P1, \{T2, \langle T4, T5\rangle\}\rangle$
12. **T5** $\langle S, T1, P1, \{\langle T4, T5\rangle, \langle T2, T3\rangle\}\rangle$ or $\langle S, T1, P1, \{\langle T2, T3\rangle, \langle T4, T5\rangle\}\rangle$

Each of the three instances for each parallel activity will generate its own set of alternate *Effect Scenarios* the combination of which will be a complete representation of all possible outcomes if the process had executed up to and including the

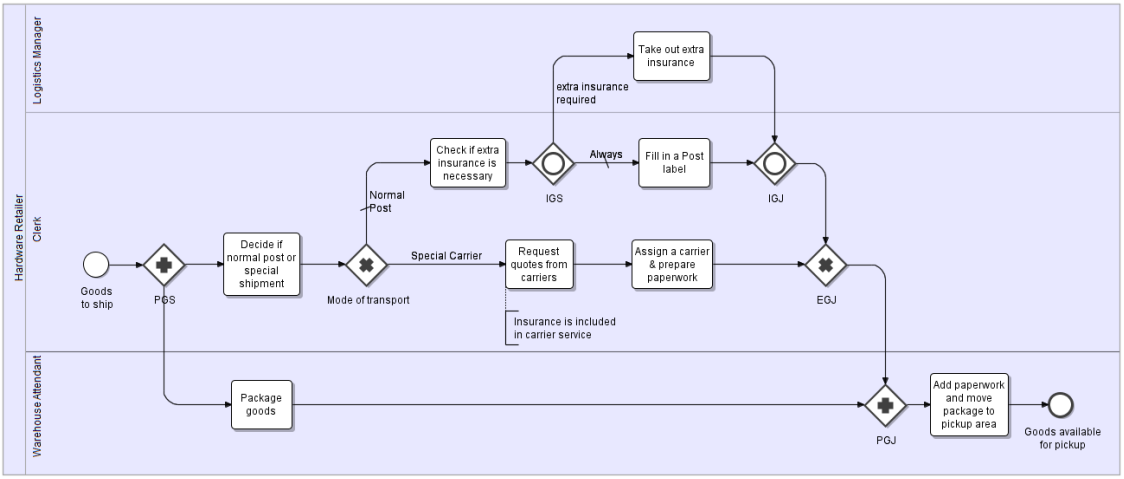

Figure 3.13: A BPMN model from the document "BPMN 2.0 by Example"[110]

selected activity. Note that *Scenario Labels* 2 and 8 are identical as are 6 and 12. At the parallel gateway join, the alternate *Effect Scenarios* are matched with their counterparts from other branches based on their origin in the *Ancestor Sequences* then assembled into *Branch Combinations*. Duplicate *Effect Scenarios* will become one during the joining procedures (see 3.8.9 for a full description of this procedure).

3.9 Example of Process Annotation and Accumulation

The BPMN model in Fig:3.13 is taken from a document "BPMN 2.0 by Example"[110]. It depicts a process in a Hardware Retail Store showing the preparatory steps required prior to shipping goods to a customer. There are three experts involved in this process: The Clerk, The Warehouse Worker and the Logistics Manger. The business analyst designing this process model will now proceed to add semantic effect annotations. We step through the annotation procedure and then show how the semantic effects are accumulated.

The first step is to identify all the artifacts that will be involved in this process. Remember that artifacts can be concepts not just material objects. Some obvious objects that stand out to the business analyst are:

```
Goods
Package
Normal Post
Special Carrier
Insurance
Post Label
Paperwork
```

These artifacts stand out because they are implied by the task labels. This is the style of reasoning a business analyst must employ using current BPMN models. The activities are the only source of information provided by the model. These artifacts are also limited by the business analyst's understanding of the process, e.g., the term "Paperwork" does not adequately describe any particular artifact. This should prompt the business analyst to seek clarification regarding the meaning of the term. After consultation with the experts involved the following additional artifacts are identified. The Clerk listed the following artifacts:

`Goods`
`Paperwork` – What is this? What does it include?
`Shipment Form` – A clarification of Paperwork.
`Purchase Order` – A clarification of Paperwork.
`Shipment Job Number`
`Shipping Database`
`Carrier`
`Transportation Method`
`Transportation Quote` – A clarification of Paperwork.
`Delivery Address`

The Warehouse Worker listed the following artifacts:

`Goods`
`Package`
`Post Label`
`Shipping Paperwork`
`Packing material`
`Packing tape`
`Cardboard Box`
`Address Label`
`Bar Code Label`
`RFID Tag`
`Fragile Stamp`
`Fragile Label`

The Logistics Manager listed the following artifacts:

`Insurance Account Number`
`Shipment Job Number`
`Shipment Form White`
`Shipment Form Yellow`

```
Database
Transportation Method
Transportation Quote
Delivery Details
Expected Delivery Time
```

Note the artifact *Paperwork* has been highlighted because it is a generic term that warrants further questioning. The Clerk has revealed that the Paperwork includes a *Shipment Form* and a *Purchase Order*. They have also revealed conceptual artifacts that represent information contained in the *Paperwork* like the *Shipment Job Number* and *Delivery Details* for example. This could be further expanded so that *Deliver Details* are broken down into address entry fields etc. but we will stick with the more generic description in this case.

These artifacts serve as a prompt for semantic effect annotation. The start event indicates that there are goods to ship. The business analyst seeks to establish the state of the artifacts in the list. There are three ways this can be done. A process model sequentially preceding this model may contribute an *Effect Scenario* that contains these states, or the business analyst may simply want to establish the initial states as facts by annotating them as *Immediate Effects* of the start event, or the business analyst may want to establish the initial states as *Conditions* of execution by annotating them to the first sequence flow in the process. In the last case, *Effect Scenarios* introduced by other process models are subject to satisfiability requirements. Annotating conditional statements to the initial sequence flow also requires establishing them as facts in the start event (*Immediate Effects*). When introducing *Effect Scenarios* from other process models, they are introduced into the *Cumulative World List* of the start event so they override the *Immediate Effects*. In this example the simplest of the three methods is used, annotating the start event with *Immediate Effects* to establish the initial states of the artifacts as assertions.

The following *Effect Scenarios* have been translated from entries in an OWL ontology into first order predicate logic. OWL provides a boolean data value for data properties but this does not affect reasoning in the way you might expect. For example, `isFragile(goods,false)` is not the negation of `isFragile(goods,true)`. Both these statements are true and can legitimately coexist in the same *Effect Scenario* without causing an inconsistency. If we choose to use boolean values in *Effects* then the negation of `isFragile(goods,true)` is ¬ `isFragile(goods,true)`. The start event has two *Effect Scenarios* annotated. To avoid repetition the first contains all *Effects* while the second only contains *Effects* that are different from the first.

Step 2: Begin Annotating Effects
Start Event: *Goods to ship*
Immediate Effect Scenario 1:

```
isReceived(purchaseOrder,true)
isInPossessionOf(purchaseOrder,clerk)
isInStock(goods,true)
isEnteredInto(goodsRecord,shippingDatabase)
isPrintedOn(deliveryAddress,shipmentFormCopy1)
isPrintedOn(deliveryAddress,shipmentFormCopy2)
isEnteredInto(deliveryAddress,shippingDatabase)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2)
isEnteredInto(expectedDeliveryTime,shippingDatabase)
isPartOf(shipmentFormCopy1,shippingPaperwork)
isPartOf(shipmentFormCopy2,shippingPaperwork)
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
isFragile(goods,true)
isPrintedOn(fragileStamp,shipmentFormCopy1)
isPrintedOn(fragileStamp,shipmentFormCopy2)
```

Immediate Effect Scenario 2:

```
¬ isFragile(goods,true)
¬ isPrintedOn(fragileStamp,shipmentFormCopy1)
¬ isPrintedOn(fragileStamp,shipmentFormCopy2)
```

The step of identifying affected artifacts has proven invaluable to the quality of annotated *Effects*. Artifacts like *Fragile Stamps* lead to predicates like `isFragile` and `isPrintedOn`. These terms expand the vocabulary in the background ontology. Note that the goods are assumed to be in stock and there is no check for this condition in the process. This alerts the business analyst to an unhandled exception. They could deal with this by including some form of exception handling in this process or annotate *Conditions* to the incoming sequence flow that prevent incoming *Effect Scenarios* from executing if they include `¬ isInStock(goods,true)`. There is also mention of an Expected Delivery Time. This may be a standard policy that is controlled by an assertion in the background knowledge base or it could be established directly in this *Effect Scenario* by adding the *Effect*, `hasValueInHoursLessThanOrEqualTo(expectedDeliveryTime, 24)`. As a background assertion it becomes global to all process models so whenever policy

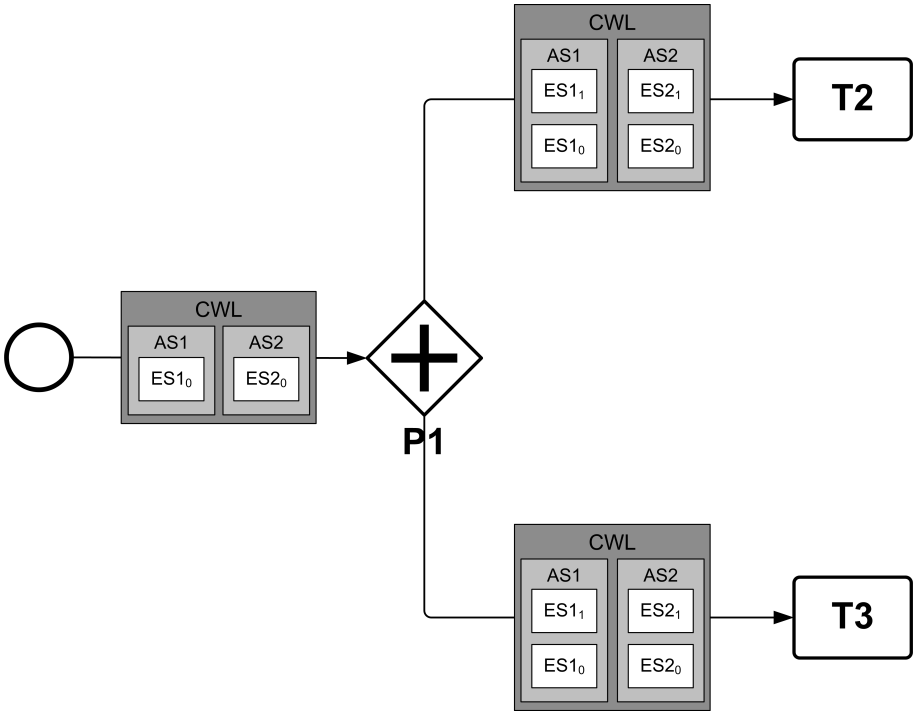

Figure 3.14: An example of how Effect Scenarios are replicated inside Ancestor Sequences and World Lists are copied to all branches when accumulated through a parallel gateway split. Ancestor sequences have been drawn vertically with the beginning at the top.

is changed then the entire process model repository is updated. In this case it is utilised as a local assertion that comes into effect when a decision needs to be made.

Each *Immediate Effect Scenario* of the start event is contained in an *Ancestor Sequence* and both *Ancestor Sequences* are contained in an *Immediate World List*. This structure can be assumed throughout this example whenever *Effect Scenarios* are referred to. When accumulated, the *Immediate World List* is replicated as the *Cumulative World List* of the start event which in turn becomes the previous *Cumulative World List* to the following parallel gateway split. The parallel gateway split has no *Immediate World List* so the previous *Cumulative World List* is replicated as the *Cumulative World List* of the parallel gateway split, with one exception. Each *Effect Scenario* is copied and added to the beginning of its *Ancestor Sequence*. Figure 3.14 shows how each *Effect Scenario* is replicated. The *Effect Scenario* $\texttt{ES1}_1$ is identical to $\texttt{ES1}_0$ and the *Cumulative World List* becomes available to the next activities on both branches. The *Effect Scenario* $\texttt{ES1}_1$ and $\texttt{ES2}_1$ are accumulated along each branch while $\texttt{ES1}_0$ and $\texttt{ES2}_0$ remain unchanged.

Moving on to the task *Decide if normal post or special shipment*, the following exclusive gateway split has two branches so this task has two *Immediate Effect Scenarios*.

Annotation

Task: *Decide if normal post or special shipment*

Immediate Effect Scenario 1:

```
hasValueInHoursLessThanOrEqualTo(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,specialCarrier)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
```

Immediate Effect Scenario 2:

```
hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
```

Accumulation of this task can be performed using *Scenario Labels* or by the real-time processing method that ignores what is happening on other parallel branches until the gateway join. Both methods are shown here because *Effect Scenarios* are minimal but as accumulation progresses through the model, the latter will only be shown for the sake of brevity. Shown first is the real-time accumulation of the task *Decide if normal post or special shipment* followed by the *Scenario Label* version of this task after describing the annotations of the *Package goods* task. With two Cumulative Effect Scenarios preceding this task and two *Immediate Effect Scenarios*, accumulation produces four *Cumulative Effect Scenarios*, shown complete. Each *Cumulative Effect Scenario* is consistent so the accumulation is simply a conjunction operation.

Accumulation in real-time

Task: *Decide if normal post or special shipment*

Cumulative Effect Scenario 1:

```
hasValueInHoursLessThanOrEqualTo(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,specialCarrier)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isReceived(purchaseOrder,true)
isInPossessionOf(purchaseOrder,clerk)
isInStock(goods,true)
isEnteredInto(goodsRecord,shippingDatabase)
isPrintedOn(deliveryAddress,shipmentFormCopy1)
isPrintedOn(deliveryAddress,shipmentFormCopy2)
isEnteredInto(deliveryAddress,shippingDatabase)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2)
isEnteredInto(expectedDeliveryTime,shippingDatabase)
```

isPartOf(shipmentFormCopy1,shippingPaperwork)
isPartOf(shipmentFormCopy2,shippingPaperwork)
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
isFragile(goods,true)
isPrintedOn(fragileStamp,shipmentFormCopy1)
isPrintedOn(fragileStamp,shipmentFormCopy2)

Cumulative Effect Scenario 2:

hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isReceived(purchaseOrder,true)
isInPossessionOf(purchaseOrder,clerk)
isInStock(goods,true)
isEnteredInto(goodsRecord,shippingDatabase)
isPrintedOn(deliveryAddress,shipmentFormCopy1)
isPrintedOn(deliveryAddress,shipmentFormCopy2)
isEnteredInto(deliveryAddress,shippingDatabase)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2)
isEnteredInto(expectedDeliveryTime,shippingDatabase)
isPartOf(shipmentFormCopy1,shippingPaperwork)
isPartOf(shipmentFormCopy2,shippingPaperwork)
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
¬ isFragile(goods,true)
¬ isPrintedOn(fragileStamp,shipmentFormCopy1)
¬ isPrintedOn(fragileStamp,shipmentFormCopy2)

Cumulative Effect Scenario 3:

hasValueInHoursLessThanOrEqualTo(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,specialCarrier)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isReceived(purchaseOrder,true)
isInPossessionOf(purchaseOrder,clerk)
isInStock(goods,true)
isEnteredInto(goodsRecord,shippingDatabase)
isPrintedOn(deliveryAddress,shipmentFormCopy1)

isPrintedOn(deliveryAddress,shipmentFormCopy2)
isEnteredInto(deliveryAddress,shippingDatabase)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2)
isEnteredInto(expectedDeliveryTime,shippingDatabase)
isPartOf(shipmentFormCopy1,shippingPaperwork)
isPartOf(shipmentFormCopy2,shippingPaperwork)
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
¬ isFragile(goods,true)
¬ isPrintedOn(fragileStamp,shipmentFormCopy1)
¬ isPrintedOn(fragileStamp,shipmentFormCopy2)

Cumulative Effect Scenario 4:

hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isReceived(purchaseOrder,true)
isInPossessionOf(purchaseOrder,clerk)
isInStock(goods,true)
isEnteredInto(goodsRecord,shippingDatabase)
isPrintedOn(deliveryAddress,shipmentFormCopy1)
isPrintedOn(deliveryAddress,shipmentFormCopy2)
isEnteredInto(deliveryAddress,shippingDatabase)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2)
isEnteredInto(expectedDeliveryTime,shippingDatabase)
isPartOf(shipmentFormCopy1,shippingPaperwork)
isPartOf(shipmentFormCopy2,shippingPaperwork)
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
isFragile(goods,true)
isPrintedOn(fragileStamp,shipmentFormCopy1)
isPrintedOn(fragileStamp,shipmentFormCopy2)

Annotating the task *Package goods* involves a lot of artifacts. The *Effects* that are common to both *Effect Scenarios* are grouped to avoid repetition. When this model was first annotated, the consulted experts had only reported that there was a Shipment Form involved in the process. During annotation the business analyst

could see that the Warehouse Worker could not work in parallel without the Shipping Form. Upon further questioning the experts explained that there were two copies of the Shipment Form that they had overlooked in their explanation.

Task: *Decide if normal post or special shipment*

Immediate Effect Scenario Common to both:

```
isSealedWith(cardboardBox,packingTape)
isSealed(cardboardBox,true)
hasPackageType(package,cardboardBox)
hasContents(package,goods)
isPrintedOn(deliveryAddress,addressLabel)
isAffixedTo(addressLabel,package)
isPartOf(addressLabel,shippingPaperwork)
isPartOf(barcode,RFIDtag)
isAffixedTo(RFIDtag,package)
isEnteredInto(barcode,shippingDatabase)
isAffixedTo(barcodeSticker,shipmentFormCopy2)
```

Immediate Effect Scenario 1:

```
isFragile(goods,true)
isAffixedTo(fragileLabel,package)
hasContents(cardboardBox,packMaterial)
hasContents(packMaterial,goods)
```

Immediate Effect Scenario 2:

```
¬ isFragile(goods,true)
hasContents(cardboardBox,goods)
```

The obvious problem here is that the business analyst has specified two *Immediate Effect Scenarios* based on whether the goods are fragile or not. There is only one activity to package goods yet clearly two are needed, i.e., *Package goods* and *Package fragile goods*. These two activities should be preceded by an exclusive gateway. For the purpose of this example only the existing model will be considered, disregarding the possibility that the goods may be fragile, with the common *Effects* excluded to conserve space. Real-time processing will only accumulate the *Cumulative Effect Scenarios* from the parallel gateway with the *Immediate Effect Scenarios* from the *Package goods* task resulting in the following *Cumulative Effect Scenarios*. Remember the fragile goods case has been disregarded.

Task: *Package goods*

Cumulative Effect Scenario 2:

isReceived(purchaseOrder,true)
isInPossessionOf(purchaseOrder,clerk)
isInStock(goods,true)
isEnteredInto(goodsRecord,shippingDatabase)
isPrintedOn(deliveryAddress,shipmentFormCopy1)
isPrintedOn(deliveryAddress,shipmentFormCopy2)
isEnteredInto(deliveryAddress,shippingDatabase)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1)
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2)
isEnteredInto(expectedDeliveryTime,shippingDatabase)
isPartOf(shipmentFormCopy1,shippingPaperwork)
isPartOf(shipmentFormCopy2,shippingPaperwork)
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
¬ isFragile(goods,true)
¬ isPrintedOn(fragileStamp,shipmentFormCopy1)
¬ isPrintedOn(fragileStamp,shipmentFormCopy2)
hasContents(cardboardBox,goods)

Working with *Cumulative Effect Scenario 2* of the *Package goods* task a *Scenario Label* accumulation of the *Decide if normal post or special shipment* task is now considered. The *Scenario Labels* for this task are first computed. Let S be the start event. Let P1 be the parallel gateway split. Let D be the decision task *Decide if normal post or special shipment* task and let PG be the *Package goods* task.

⟨S,P1,D⟩
⟨S,P1,{D,PG}⟩

Note that there are only two *Scenario Labels* for task D. The *Scenario Label* generation method used in [63] would have produced the following three paths:

⟨S,P1,D⟩
⟨S,P1,D,PG⟩
⟨S,P1,PG,D⟩

Using the new joining algorithms, the last two *Scenario Labels* result in exactly the same outcome so there is no need to compute them separately. Let us now look at how the first set of two *Scenario Labels* applies to the accumulation of *Effect Scenarios*. There exists a set of four *Cumulative Effect Scenarios* for the first

Scenario Label accumulated from task D using the real-time method . The second *Scenario Label* indicates a join between tasks D and PG.

The first step in joining involves *Clustering* (see chapter 3.8.11). *Cumulative Effect Scenarios* on a single branch that share the same *Effect Histories* are collected into *Branch Clusters*. In this case the *Branch Cluster* for task PG is identical to its *Cumulative World List* because only a single *Cumulative Effect Scenario* is being considered. Task D on the other hand has four *Cumulative Effect Scenarios* that have originated from two. *Clustering* will produce the following *Clusters*. Note the reference to the *Ancestor Sequences* containing the *Cumulative Effect Scenarios* described earlier.

Task: *Decide if normal post or special shipment*
Cluster 1 includes:
D *Ancestor Sequence 1*
D *Ancestor Sequence 4*
Cluster 2 includes:
D *Ancestor Sequence 2*
D *Ancestor Sequence 3*

Task: *Package goods*
Cluster 1 includes:
PG *Ancestor Sequence 2*

The *Clusters* are then collected into the following *Branch Groups* (see chapter 3.8.12). Recall that each *Branch Group* for a parallel gateway join contains exactly one *Cluster* from each branch and all *Ancestor Sequences* inside the *Clusters* of each *Branch Group* share the same *Effect Scenario History*.

Group 1 includes:
D *Cluster 1*
PG *Cluster 1*
Group 2 includes:
D *Cluster 2*
PG *Cluster 1*

Ancestor Sequences in *Branch Group 1* have a distinctly different *Effect Scenario History* to those in *Branch Group 2*. The *Ancestor Sequences* in the *Clusters* are then collected into *Branch Combinations*, one *Ancestor Sequence* from each *Cluster* in a *Branch Group*.

Branch Combination 1 includes:

D *Cluster 1*

D *Ancestor Sequence 1*

PG *Cluster 1*

PG *Ancestor Sequence 2*

Branch Combination 2 includes:

D *Cluster 1*

D *Ancestor Sequence 4*

PG *Cluster 1*

PG *Ancestor Sequence 2*

Branch Combination 3 includes:

D *Cluster 2*

D *Ancestor Sequence 2*

PG *Cluster 1*

PG *Ancestor Sequence 2*

Branch Combination 4 includes:

D *Cluster 2*

D *Ancestor Sequence 3*

PG *Cluster 1*

PG *Ancestor Sequence 2*

Once again the complexity has been reduced by showing only one *Branch Combination* accumulation. Working with *Branch Combination 3* there are two *Ancestor Sequences*, D *Ancestor Sequence 2* and PG *Ancestor Sequence 2*. The first *Cumulative Effect Scenario* of each *Ancestor sequence* may contain propagated *Effects* from the second *Effect Scenario* in the *Ancestor Sequence*, i.e., they have carried through from the beginning of the parallel gateway split. These need to be removed from the first *Effect Scenarios* in the *Ancestor Sequences*. The ♦ is used to indicate a propagated *Effect* from the second *Effect Scenario* in the *Ancestor Sequence*.

Task: *Decide if normal post or special shipment*

Cumulative Effect Scenario 2:

```
hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isReceived(purchaseOrder,true) ♦
```

isInPossessionOf(purchaseOrder,clerk) ♦
isInStock(goods,true) ♦
isEnteredInto(goodsRecord,shippingDatabase) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy1) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy2) ♦
isEnteredInto(deliveryAddress,shippingDatabase) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2) ♦
isEnteredInto(expectedDeliveryTime,shippingDatabase) ♦
isPartOf(shipmentFormCopy1,shippingPaperwork) ♦
isPartOf(shipmentFormCopy2,shippingPaperwork) ♦
isInPossessionOf(shipmentFormCopy1,clerk) ♦
isInPossessionOf(shipmentFormCopy2,warehouseAttendant) ♦
¬ isFragile(goods,true) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy1) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy2) ♦

Task: *Package goods*

Cumulative Effect Scenario 2:

isReceived(purchaseOrder,true) ♦
isInPossessionOf(purchaseOrder,clerk) ♦
isInStock(goods,true) ♦
isEnteredInto(goodsRecord,shippingDatabase) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy1) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy2) ♦
isEnteredInto(deliveryAddress,shippingDatabase) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2) ♦
isEnteredInto(expectedDeliveryTime,shippingDatabase) ♦
isPartOf(shipmentFormCopy1,shippingPaperwork) ♦
isPartOf(shipmentFormCopy2,shippingPaperwork) ♦
isInPossessionOf(shipmentFormCopy1,clerk) ♦
isInPossessionOf(shipmentFormCopy2,warehouseAttendant) ♦
¬ isFragile(goods,true) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy1) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy2) ♦
hasContents(cardboardBox,goods)

Removing the propagated Effects leaves only the Effects that have been introduced by parallel activities.

Task: *Decide if normal post or special shipment*
Cumulative Effect Scenario 2:

```
hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
```

Task: *Package goods*
Cumulative Effect Scenario 2:

```
hasContents(cardboardBox,goods)
```

As should be the case, the union of these two *Cumulative Effect Scenarios* is consistent with the background knowledge base. Unions of *Cumulative Effect Scenarios* in *Branch Combinations 1 and 2* will result in inconsistent *Effect Scenarios*, indicating the erroneous design of this process model. The union of the above two *Cumulative Effect Scenarios* produces the following *Effect Scenario*.
Immediate Effect Scenario of the Join:

```
hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
hasContents(cardboardBox,goods)
```

The above unioned *Effect Scenario* becomes the *Immediate Effect Scenario* of the join, indicating that the parallel activities will have caused these outcomes had they both been executed in any order. This *Immediate Effect Scenario* is now accumulated with the *Effect Scenario* that is second in the *Ancestor Sequence*, shown again here for convenience.
Parallel Gateway Split:
Cumulative Effect Scenario 2:

```
isReceived(purchaseOrder,true) ♦
isInPossessionOf(purchaseOrder,clerk) ♦
isInStock(goods,true) ♦
isEnteredInto(goodsRecord,shippingDatabase) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy1) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy2) ♦
isEnteredInto(deliveryAddress,shippingDatabase) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1) ♦
```

isPrintedOn(expectedDeliveryTime,shipmentFormCopy2) ♦
isEnteredInto(expectedDeliveryTime,shippingDatabase) ♦
isPartOf(shipmentFormCopy1,shippingPaperwork) ♦
isPartOf(shipmentFormCopy2,shippingPaperwork) ♦
isInPossessionOf(shipmentFormCopy1,clerk) ♦
isInPossessionOf(shipmentFormCopy2,warehouseAttendant) ♦
¬ isFragile(goods,true) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy1) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy2) ♦

Note that all these *Effects* were the propagated *Effects* removed from the first *Effect Scenarios* in the *Ancestor Sequences*. When these Effects are added to the *Immediate Effect Scenario* for the join, the resulting *Cumulative Effect Scenario* for the join is consistent and no *Immediate Effects* over-ride any of the *Effects* in the second *Effect Scenario* in the *Ancestor Sequence*. Accumulation in this case is a simple union.

The Joining of {D,PG}:

Cumulative Effect Scenario from Branch Combination 3:

hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
hasContents(cardboardBox,goods)
isReceived(purchaseOrder,true) ♦
isInPossessionOf(purchaseOrder,clerk) ♦
isInStock(goods,true) ♦
isEnteredInto(goodsRecord,shippingDatabase) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy1) ♦
isPrintedOn(deliveryAddress,shipmentFormCopy2) ♦
isEnteredInto(deliveryAddress,shippingDatabase) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy1) ♦
isPrintedOn(expectedDeliveryTime,shipmentFormCopy2) ♦
isEnteredInto(expectedDeliveryTime,shippingDatabase) ♦
isPartOf(shipmentFormCopy1,shippingPaperwork) ♦
isPartOf(shipmentFormCopy2,shippingPaperwork) ♦
isInPossessionOf(shipmentFormCopy1,clerk) ♦
isInPossessionOf(shipmentFormCopy2,warehouseAttendant) ♦
¬ isFragile(goods,true) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy1) ♦
¬ isPrintedOn(fragileStamp,shipmentFormCopy2) ♦

At this point it looks as though only {D,PG} of the second *Scenario Label* ⟨S,P1,{D,PG}⟩ has been accumulated but in fact the entire *Scenario Label* as been accumulated because the preceding steps have already been completed to arrive at the previous *Cumulative Effect Scenario* that was second in the *Ancestor Sequence*. This *Scenario Label* technique computes all possible outcomes for any activity within a parallel *Gateway Structure*.

If this technique is compared to the two sequential path *Scenario Labels*

⟨S,P1,D,PG⟩

⟨S,P1,PG,D⟩

it can be seen that PG is allowed to undo *Effects* that result from D or D is allowed to undo *Effects* that result from PG. When there are no conflicting *Effects* on different branches then these two sequences will yield the same *Effect Scenario*. The method will require further algorithms to detect and remove duplicate *Effect Scenarios*. However, this method will not detect conflicting *Effects* on different branches so erroneously designed process models can go unchecked.

Now consider an accumulation that results in an inconsistent *Effect Scenario* using the real-time method of accumulation. Let's look at the situation where the clerk receives both *Shipment Forms* at the beginning of the process then passes the second copy onto the warehouse attendant. This task is not included in this process so it was assumed that it had already been established prior to the process commencing. Suppose that as a result of *Package goods*, the warehouse attendant is in possession of shipmentFormCopy2. The previous *Cumulative Effect Scenario* from the parallel gateway split has been limited to only two *Effects*.

```
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,clerk)
```

An *Immediate Effect Scenario* of *Package goods* would contain the *Effect*:

```
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
```

Given these two *Effect Scenarios* the *acc*( ) function (see appendix A.4) would first union them to produce:

```
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
```

This is then combined with the background knowledge base (KBR) and tested for consistency and it fails because there exists a rule in KBR

$\forall x\ isInPossessionOf(x,y) \rightarrow \neg isInPossessionOf(x,z)\ where\ y \neq z$

The *acc*( ) function would then pass the inconsistent *Effect Scenario* to the possible

worlds function (see appendix A.3) which in turn will pass it on to the combinatorial function (see appendix A.2). This function will return the following possible worlds:

Possible World 1:

```
isInPossessionOf(shipmentFormCopy1,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
```

Possible World 2:

```
isInPossessionOf(shipmentFormCopy2,clerk)
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
```

Note that the second *Effect* showing the Form in possession of the warehouse attendant is present in both *Effect Scenarios* because it is an *Immediate Effect* which takes priority over all previous *Cumulative Effects*. Previous *Cumulative Effects* are added in all combinations but all *Immediate Effects* will exist in every possible world. Possible World 2 is clearly inconsistent with the background rule. Possible World 1 is consistent so the *pw*( ) function will stop asking the combinatorial function for subsets of this *Effect Scenario*. It will, however, pass back Possible World 2 to the combinatorial function which will return the following subset:

Possible World 3:

```
isInPossessionOf(shipmentFormCopy2,warehouseAttendant)
```

The *pw*( ) function will first test if Possible World 3 is a subset of any existing consistent *Effect Scenarios*, i.e., Possible World 1. It is a subset so consistency checking can be dispensed with and Possible World 3 is rejected. Possible World 1 is returned as the only *Cumulative Effect Scenario* that results from the *Package Goods* task.

Let's now look at the exclusive gateway split preceding the task, *Check if extra insurance is necessary*. Let `C` represent task *Check if extra insurance is necessary*. First, consider the four *Effect Scenarios* generated by the task *Decide if normal post or special shipment*. Only the key *Effects* have been included to save space.

Task: *Decide if normal post or special shipment*

Cumulative Effect Scenario 1:

```
hasValueInHoursLessThanOrEqualTo(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,specialCarrier)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isFragile(goods,true)
isPrintedOn(fragileStamp,shipmentFormCopy1)
```

isPrintedOn(fragileStamp,shipmentFormCopy2)

Cumulative Effect Scenario 2:

hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
¬ isFragile(goods,true)
¬ isPrintedOn(fragileStamp,shipmentFormCopy1)
¬ isPrintedOn(fragileStamp,shipmentFormCopy2)

Cumulative Effect Scenario 3:

hasValueInHoursLessThanOrEqualTo(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,specialCarrier)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
¬ isFragile(goods,true)
¬ isPrintedOn(fragileStamp,shipmentFormCopy1)
¬ isPrintedOn(fragileStamp,shipmentFormCopy2)

Cumulative Effect Scenario 4:

hasValueInHoursGreaterThan(expectedDeliveryTime,24)
hasModeOfTransport(deliveryMethod,normalPost)
isPrintedOn(deliveryMethod,shipmentFormCopy1)
isFragile(goods,true)
isPrintedOn(fragileStamp,shipmentFormCopy1)
isPrintedOn(fragileStamp,shipmentFormCopy2)

These four *Cumulative Effect Scenarios* are propagated as the *Cumulative Effect Scenarios* of the exclusive gateway split. Before they can be utilised as a previous *Cumulative Effect Scenario* for task C they are subjected to a *Condition* check. The sequence flow between the exclusive gateway split and task C contains an annotated *Condition Scenario*. The business analyst has identified which *Conditions* must be in existence for task C to execute. The *Condition Scenario* contains only one *Condition*:

hasModeOfTransport(deliveryMethod,normalPost)

Cumulative Effect Scenarios 2 and 4 satisfy the *Condition* because they contain this *Condition* so they become the only two previous *Cumulative Effect Scenarios* for task C. These are now accumulated with task C's *Immediate Effect Scenarios*.

The same technique is used for the inclusive gateway except that *Cumulative Effect Scenarios* can be propagated along more than one branch depending on whether they satisfy the *Condition Scenarios* of more than one branch.

In this example it has been shown how the act of annotating *Effects* to a process model can reveal design problems that may not have been implicitly obvious. This shows how semantic effect annotation of business process models can enhance the quality of process design. The practical application of the principal accumulation algorithms has also been demonstrated.

3.10 Summary

This chapter presents the conceptual underpinnings of the ProcessSEER framework. It explores the basic elements of a process, i.e., actions, conditions and effects, as well as different approaches to represent them. Formal definitions of the elements are provided that make up the ProcessSEER framework. Process model notations include a wide variety of symbols. The focus is on the most commonly used process elements and devised algorithms for accumulating effects over different combinations of these elements. Each of these algorithms is discussed in reference to Appendix A. This chapter is concluded with an example of the framework being utilised in a logistical setting.

Chapter 4

Clinical Applications

4.1 Introduction

The notion of *care flow management* [115] has become the focus of considerable research attention in the recent past. It builds on the premise that process management principles and techniques can deliver value in clinical settings as much as it delivers value in settings such as business process management. Clinical process management can help encode clinical guidelines which can provide a reference baseline for clinicians. These can leverage the coordination capabilities of Workflow Management Systems (WfMS) in ensuring that treatment steps are executed correctly relative to reference guidelines. More generally, care flow management also addresses the administrative aspects of health care, both from the perspective of health care providers and patients [34].

Existing techniques/notations for modelling processes, such as the industry-standard Business Process Modelling Notation (BPMN) [112], only model the coordination semantics of processes, but offer no facility for describing the effects of processes (or steps/activities within processes). Thus, we are able to clearly specify the required sequencing of activities, for instance, but cannot specify the effects that these activities would have on the domain/context in which the process would execute (beyond the minimal information that can be conveyed via the nomenclature of tasks). However, these effect descriptions are critical in determining whether process designs have been correctly formulated. Additionally, much of the analysis required for process compliance management [51, 55], change management [80], enterprise process architectures [82] and management of a business process life cycle [83] relies on being able to refer to the *effect semantics* of business processes.

The detection of treatment conflicts between multiple treatment protocols that are co-incident (on the same patient) is a difficult and open problem that is particularly exacerbated in the context of treatment of multiple medical conditions

co-occurring in aged patients. For example, a clinical protocol for prostate cancer treatment requires the administration of androgen-suppressing medication. This could negatively interact with another, co-incident, protocol, if the same patient were being treated for renal disease via haemodialysis, where androgen-enhancers are frequently administered. Treatment conflicts such as these are subtle, and usually difficult to detect using automated means. Traditional approaches to clinical decision support would require significant amounts of clinical knowledge to be acquired and encoded, with the well-known difficulties associated with large-scale knowledge acquisition.

In this chapter, the ProcessSEER framework and supporting tool [63] are leveraged to provide a practitioner-accessible means for providing semantic effect annotations of process models to deliver value in clinical process management in a range of different ways. The focus is primarily on the use of this machinery in detecting *treatment conflicts* between co-incident treatment protocols, i.e., situations where the application of a treatment protocol on a patient contra-indicates the application of another treatment protocol on the same patient. These are often subtle, and otherwise difficult to detect using automated means. It is also argued that semantic effect annotations of treatment protocols can be leveraged for a variety of other tasks, including identifying instances where different specialists arrive at differing interpretations of the same protocol, as well as pedagogical applications. The chapter also includes a case study of using the ProcessSEER tool in a clinical environment that revealed some valuable insights into human computer interaction.

4.2 Detecting inter-process interactions

The ProcessSEER framework utilises two methods of accumulating *Effect Scenarios*. The first method involves the use of *Scenario Labels* [52]. These effectively describe the paths taken through a process model to obtain the corresponding *Effect Scenarios* (as a sequence of gateway and activity identifiers). This is important given that it is possible to arrive at a given task in a process model via multiple alternative paths. The method will generate all possible outcomes for any point in the process answering the question, "what would have happened if the process had executed up to this point?" (see chapter 3.6.1). Answering this question comes at a cost. Nested parallel and exclusive *Gateway Structures* can cause an exponential increase in the number of *Scenario Labels* generated by a process model. The second method walks through the model like a group of individual agents and adopts the attitude that an agent on one branch of a parallel *Gateway Structure* does not know what agents on other parallel branches are doing until they meet at a parallel gateway join.

The second method minimises processing time without compromising the integrity of the accumulated outcomes (see chapter 3.6).

The start event of a BPMN model is a doorway into the model. A patient's records will contain information about treatments they are currently receiving or have received in the past. Using a repository of clinical process models a clinician can access these processes without violating patient confidentiality because the process models do not refer to a patient directly. The *Effect Scenarios* of the patient's existing treatment protocols can be plugged into the treatment protocol that the clinician is intending to prescribe. When those *Effect Scenarios* are accumulated into the intended treatment protocol, any conflicts that occur will be highlighted. It is foreseeable that patient records sometime in the future could contain Effects (written in a universal standard) that could be plugged into any intended treatment protocol for an instant evaluation of potential side-effects or unexpected outcomes. The ProcessSEER framework makes this type of evaluation possible.

Conflicts can be detected in two ways. First, when *Effect Scenarios* cannot be propagated to protocol activities because they do not satisfy *Condition Scenarios* annotated to the outgoing flows from each OR- or XOR-split gateway preceding the activity. Second, when rules in the background knowledge base are violated causing inconsistencies in the resulting *Effect Scenarios*. These *interaction-checking mechanisms* help clinicians identify situations where certain treatment steps should not logically co-occur in an instance of the same patient, because of contradictory conditions (e.g., a treatment step that applies only to diabetics vs. one that applies only to non-diabetics).

Plugging the *Effect Scenarios* from one treatment protocol into another involves taking the *Cumulative Effect Scenarios* from the end event of one process model and adding them to the start event of the intended treatment process model. It may not be necessary to plug every *Cumulative Effect Scenario* into the intended treatment model so if the clinician is aware of a particular outcome pertaining to the patient then that can be chosen as the input state in isolation to any others. During accumulation these plugged-in *Effects* will interact with the *Effects* in the intended treatment process generating *conflict flags* if a potential conflict is detected between the processes.

In many cases the start event can often have no *Immediate World List* annotated to it so the previous process model's end event *Cumulative Effect Scenarios* become the *Cumulative Effect Scenarios* of the start event in the intended process model. When *Immediate Effects* are annotated to a start event, they act like a security guard for the entire process. They establish a state of the world that must be in effect at the beginning of the process. In the case where the start event of an intended procedure contains an *Immediate World List* then the *Cumulative Effect*

Scenarios from the existing patient treatment protocol are accumulated with the *Immediate Effect Scenarios* of the start event. This first accumulation may reveal conflicts through inconsistent outcomes or violation of some background rule thus requiring no further computation.

The ProcessSEER framework was extended to address the problem of detecting interactions between co-incident clinical protocols. A distinction between *mandatory effects* and *potential effects* was introduced. Such a distinction is common in clinical settings, for instance, between the (mandatory) intended effects of medication, and (potential) side-effects/complications. It is assumed that a *patient-specific knowledge base* ($P - KB$) exists as well as a *background knowledge base* (KBR), but the techniques presented here can be of use even when these are empty (as the example in the next section illustrates). The *interaction-checking mechanism* takes as input a set of clinical process models of existing patient treatment protocols, a clinical process model of an intended treatment protocol, and a $P - KB$. It returns a set of *conflict flags* if a potential conflict is detected between the input processes. In the following, a *conflict* refers to situations where $\{es_1, \ldots, es_n\} \cup P - KB \cup KBR$ is inconsistent, where each es_i represents an *Effect Scenario* obtained from a distinct process that is part of the input. Note that when conflict checking must be restricted to mandatory (resp. potential) effects, these components can be directly extracted from an *Effect Scenario* (since each assertion in an *Effect Scenario* is thus labelled). Conflict flags can be of various kinds:

- *Strong conflicts:* These involve situations where all *Effect Scenarios* associated with a task in one process conflict with all the *Effect Scenarios* associated with a task in a distinct process.

- *Weak conflicts:* These involve situations where some (but not all) *Effect Scenarios* associated with a task in one process conflict with some (but not all) *Effect Scenarios* associated with a task in a distinct process.

- *Mandatory-Mandatory (MM) effect conflicts:* These involve situations where the mandatory effects within an *Effect Scenario* associated with a task in one process conflict with the mandatory effects in an *Effect Scenario* associated with a task in a distinct process.

- *Mandatory-Potential (MP) effect conflicts:* These involve situations where the mandatory effects within an *Effect Scenario* associated with a task in one process conflict with the potential effects in an *Effect Scenario* associated with a task in a distinct process.

- *Potential-Potential (PP) effect conflicts:* These involve situations where the potential effects within an *Effect Scenario* associated with a task in one process conflict with the potential effects in an *Effect Scenario* associated with a task in a distinct process.

These categories provide a rich vocabulary for describing conflicts and are not mutually exclusive (it is possible to obtain a strong MP conflict or a weak PP conflict).

The following section provides one substantive example of the detection of treatment conflicts via semantically annotated process models. Even with this setting, the size of the process models, and the effect annotations makes it impossible to display the models in their entirety. The conflict flag obtained in this instance is a *strong MM conflict*.

4.3 A Clinical Example

A large-scale exercise in process modelling (in the BPMN notation) of clinical protocols was initiated for this research program. A total of 55 cancer trial protocols and 26 clinical pathway protocols for a breast cancer patient have been modelled in BPMN. The resulting models are large, and after semantic effect annotation, larger still. Fig:4.2 describes a small portion of a prostate cancer trial protocol modelled in BPMN.

The effect annotation of the task "Adjust dosage/schedule" in Fig:4.2 proceeds as follows (only the natural language version is provided and omits the formal version obtained via ontological markup): *If there was evidence of bowel toxicity then administer constipating pain relief. This may contra-indicate medication having diarrhoea as a side effect. If there is increasing fatigue in the patient then a full blood count should be requested.*

As part of the treatment of prostate cancer it is common to prescribe anti-androgen medication to reduce androgen levels in the patient's blood. In (Fig:4.1) the action of prescribing an anti-androgen (Flutamide) is represented by a task/action icon in a BPMN model. *Effect Scenarios* from the patient treatment process model will contain *Effects* that register the prescribed Flutamide.

A common step in a renal haemodialysis protocol (full BPMN model omitted here due to space restrictions) is the prescription of androgen-enhancing medication. When the haemodialysis protocol *Effect Scenarios* are plugged into the prostate cancer protocol, this would clearly generate a conflict flag using the machinery described in the previous section.

The most compelling motivation for semantic effect annotation of clinical process models is the detection of treatment conflicts, but there are other useful applications

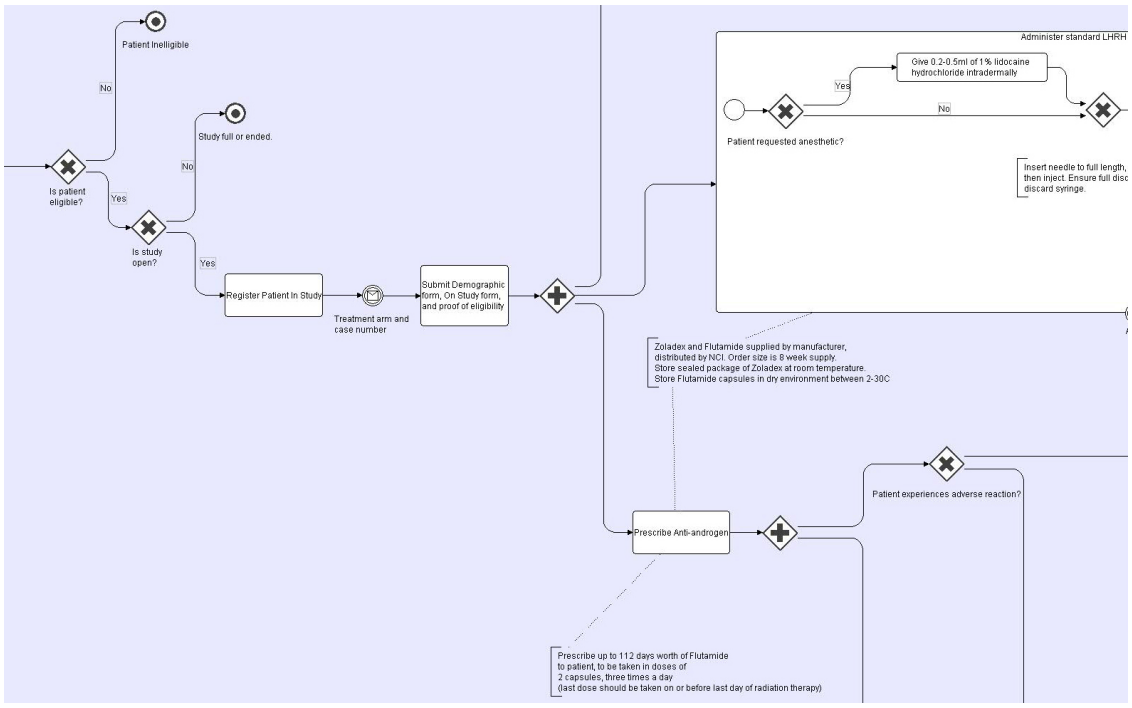

Figure 4.1: A section from a prostate treatment BPMN process model showing the prescription of anti-androgen medication.

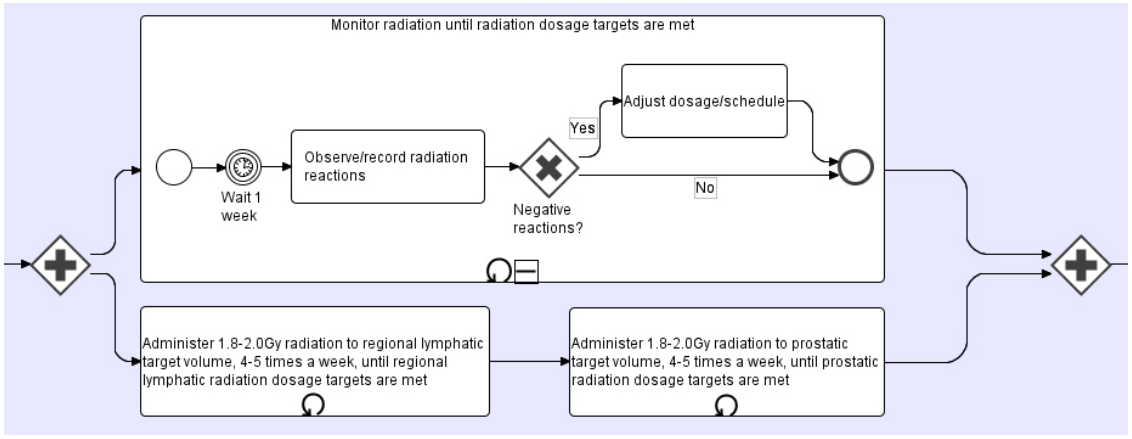

Figure 4.2: A section from a prostate treatment BPMN process model showing a sub process for monitoring radiation treatment.

for this machinery as well. A BPMN model explains WHAT to do, but rarely HOW to do it (in any significant detail) or WHY an action is being performed. Semantic effect annotations added to a process model provide a mechanism for describing the HOW in considerably greater detail. Based on preliminary experience, this can help identify situations where specialists agree on (the broad picture of) a clinical protocol, but disagree on the detail of its implementation. Semantic effect annotations can help in answering the WHY question, which has pedagogical applications. They also have applications in the Workflow Management Systems (WfMS) space for controlling task allocation. In the following case study clinicians expressed enthusiasm, with a healthy amount of skepticism, for a WfMS that could dynamically respond to the continual change in operational environments.

4.4 Clinical Case Study

This is a case study about process modelling in an oncology ward. The study identified a number of specific roles in the knowledge elicitation process. It also identified some fundamental problems with the format of semantic effect annotations and prompted a change in the way questions were asked to elicit information. A Care Flow project was undertaken for the Illawarra Cancer Care Clinic (ICCC) to model the processes involved in the care of a patient with breast cancer. The models were to cover all aspects of the patient's journey through the medical system, from diagnosis to treatment. The collection of data was performed manually and required consultation with different clinicians including physicians, pharmacists, pathologists, surgeons and oncologists as well as nursing and administrative staff. The clinical processes were modelled using the Business Process Modelling Notation (BPMN).

The clinical pathway of a breast cancer patient was originally modelled at a very high level. A sizeable wall in one of the corridors at the ICCC was devoted to mapping this clinical pathway. The idea of using the wall as a canvas for the entire patient journey has its origins in agile software development techniques [138]. Clinicians and staff, who were directly involved in the care of a breast cancer patient, were encouraged to stick Post-It notes to this wall describing the different stages and their involvement in the patients treatment. Similar techniques have been adopted in [38]. Gradually over time the entire clinical pathway took shape. The exercise proved to be an invaluable tool in the collection of data from employees in a working environment. Participants in this exercise commented on the effectiveness of mapping on the wall because it allowed them to see and understand how their efforts contributed to overall patient care.

Previous attempts, in other working environments, to gather such information from employees, through direct questioning, were met with suspicion. Individual

interviews tended to focus the interviewee's attention on themselves leading to a reported sense that "they were being scrutinised". This hampered knowledge acquisition because of their reluctance to divulge details about their working practices. The shift in focus away from the employee onto the patient's care removed the stigma surrounding the direct questioning approach. Both clinical and administrative staff were found to be enthusiastic about contributing their knowledge to this larger picture portrayed on the wall. Having this oversight of the entire process encouraged participation. The one-on-one interview environment was still employed but only after the wall map had been completed. Interviewees, by that time, had become comfortable with the information gathering exercise. Perhaps the greatest benefit of utilising the wall was that employees could see how their information was being used. It also allowed them to visualise how their contribution fitted into the bigger picture of patient care.

The focus of the wall was on the patient's journey through the health care system. The clinical pathway did not limit itself to the patient's journey through the ICCC. It covered all aspects of the journey, including a patient's first consultation with their general practitioner, pathology testing, various forms of treatment and finally recovery. The data from the wall was entered into a spreadsheet which then became the basis for inquiry when modelling the clinical processes. Employees were further reassured during interviews that the information they were providing was not only being used for the improvement of patient care but also for understanding and recognition of the importance of each individual's contribution as a care provider. This simple acknowledgement greatly improved employee cooperation during the data gathering process.

The role of the interviewer requires these interpersonal skills to maximise the quality and quantity of information gathered. However, it was found to be an impediment to communication if the interviewer had knowledge of the required format of the information. For example, if the interviewer is asking the interviewee to disambiguate their description of an activity or effect then the natural flow of communication is interrupted. This became obvious when the interviewer was a process analyst that understood the requirements of ontology development. The analyst would constantly seek clarification causing the employee to lose focus while trying to rephrase their description. The interviewee soon became disinterested and the quality of information deteriorated. In contrast, an interviewer with no knowledge of the semantic effect annotation format collected an abundance of information. It was therefore concluded that the Interviewer should be a distinct role separate from the business analyst.

4.5 Information Gathering

A direct correlation was found between the focus of attention on the role being examined and the willingness of employees to divulge information about their jobs. For example, if the process analyst focused on the role of the employee and the tasks they performed, the employee reported feeling scrutinized and would be reluctant to divulge task related details. However, if the process analyst focused on the role of the patient/customer and the careflow provided, employees became enthusiastic about explaining what they did. In the first case employees were asked what they did which evoked suspicion. In the second case employees were asked what they contributed to the wellbeing of another person (the patient/customer). This invoked a sense of pride and focused on their positive contribution and most importantly, their worth to the organisation.

Patient focused inquiry improved employee participation in the program but interviews were still isolated. Police use isolated interviews to interrogate suspects in a crime. Any association with interrogation was counterproductive to the interview process. A more productive association was to expand each employees perception of their individual contribution to the care of a patient. This was achieved by devoting a sizable wall in the hospital to mapping the clinical pathway of a breast cancer patient through the medical system.

It was found that if the interviewer had knowledge about how the task information would be utilised then that would actually impede the elicitation process. Too much emphasis was placed on the correct wording of a particular task by the interviewer which interrupted the interviewee's train of thought and greatly increased the time per interview. Allowing the interviewee to elaborate in their own words provided more information and required less time per interview. It was actually found to be advantageous for the process analyst to have no understanding of knowledge engineering practices. This freed them up to communicate on an equal level with employees and supported the employee's perception of genuine inquiry from the interviewer.

4.6 Annotating Effects

Once the breast cancer life-cycle processes had been modelled, semantic effects were annotated to the tasks and events. Annotation in practice revealed a number of problems with the original approaches. When a business analyst was asked, "what is the effect of doing this task", they would generally reply with something like "the task is done". In fact many of the effects annotated in the first attempt were of this quality. Business analysts reported finding it difficult to think how or what to

describe as an effect. A new line of questioning was developed. Instead of asking the business analyst what was the effect of doing this task, they were asked what things were involved in performing the process. For each process model, the business analyst would list all the things they thought would be affected by the process. They then viewed each task or event in the model and questioned how it would affect each of those things. Naturally all things were not affected by every activity in the model but this method changed the way they thought about the effects of activities. The response to this change was positive and the quality of effect descriptions improved dramatically.

The language in which these effects need to be specified should ideally be formal, permitting sophisticated tool support for several of the analysis and reasoning tasks mentioned above. Formal languages are typically not practitioner-accessible while informal annotations make substantive tool support difficult to devise. The use of controlled natural language (CNL) [132] was chosen as a compromise between these two extremes, offering the analyst a repertoire of sentence schemas in which to describe the effects - populating a sentence schema generates a correspondingly instantiated formal annotation. To avoid placing an unduly heavy burden of annotation on analysts, the approach described here only requires that analysts provide a description of the *Immediate Effects* of each process task, i.e., a context-independent specification of the functionality (together with relevant associated ramifications) of each task. These are then accumulated into *Cumulative Effect Scenario* annotations in a context-sensitive manner, such that the Cumulative Effect Scenario annotations associated with any task in a BPMN process model would describe the effects achieved by the process were it to execute up to that point (see chapter 3). Note that such a description will necessarily be non-deterministic, i.e., there might be alternative *Effect Scenarios* that might transpire if a process has executed up to a certain point in a process model. The non-determinism stems from two sources. First, a process might have taken different paths through a process model to arrive at a certain point. Second, the effects of certain process steps might "undo" the effects of prior process steps. This is often described as the *belief update* or *knowledge update* problem - multiple alternative means of resolving the inconsistencies generated by the "undoing" of *Effects* is another source of non-determinism.

The ProcessSEER modelling tool (named after the framework) was used to model the clinical processes in BPMN. The tool also facilitates the annotation of semantic effects to the activities and events in the model. The first version of the ProcessSEER tool provided a single data entry field that accepted a sentence written in Attempto Controlled English (ACE) [43, 44]. Annotated effect sentences became more complex from business analysts trying to cram in the maximum amount of information. This prompted further research into the nature of an effect (see Chapter

3.2). The result was to limit effects to single sentences that described a single attribute or relationship of an artifact employed in the process. However, it was found that there was still a reluctance to learn ACE and thus a reluctance to annotate semantic effects to their process models.

The tool was upgraded as a consequence of these insights to provide three data entry fields, one for the artifact that was being affected, the second for the relationship or attribute of the artifact that was being changed and the third for which the artifact was related to or the value of the attribute of the artifact. This met with acceptance because it mirrored the new approach about how to think about effects. It also did not require knowledge of a CNL. Business analysts were free to enter any terms they liked within the scope of each data entry field. Discussion about how this was achieved can be found in chapter 6.

The original approach to manual semantic effect annotation placed the business analyst in an untenable position, right in the middle of communicating between experts in the field and knowledge engineers working with ontologies. In this position the business analyst needed to know both roles in addition to their own. In practice this proved to be unsuccessful. Separating the roles of Interviewer, Business Analyst and Knowledge Engineer provided a workable arrangement. The interviewer would elicit information from the expert and pass the information, written in a natural language, to the business analyst. The business analyst would then translate the text into the three field data entry of the ProcessSEER tool. If the text was difficult to understand, the business analyst could approach the expert directly for clarification. The new data entry fields meant the business analyst didn't need to know specific terminology or grammar. The knowledge engineer would make any changes that were necessary to the terms used in the effect descriptions or they would integrate new terms into the domain ontology. If the knowledge engineer had any questions about the terminology used, they could also consult directly with the expert. In this way the process model provides contextual information that prompts knowledge engineers to ask appropriate questions thus improving knowledge acquisition.

Separating these four roles significantly improved acceptance of semantically annotating effects to business process models. To recap, the four roles are:

- Expert
- Interviewer
- Business Analyst
- Knowledge Engineer

The expert is the person who performs a task in real life. They can be the person who cleans the windows or an oncologist. The interviewer needs the interpersonal skills

so as not to incite suspicion when interviewing employees and the technical skills for writing job specifications. The business analyst designs, models and analyses the processes and annotates the semantic effects. The knowledge engineer integrates the terms, used in semantic effect annotation, into a background ontology which is used to reason about the outcome of process models. There are definite stages to the knowledge elicitation process and disambiguation should only be employed after data collection.

4.7 Task Labelling

Many process models provide an observational view of tasks being performed by others. The observed behaviour of another is recorded as an activity or a sequence of activities. The important thing to note here is that the observer is interpreting what the other person is doing. The observer/instructor will typically use a label to represent a task but the detail of performing that task is held in the experience of the person performing it.

A label, for example, is something like "sharpen the pencil". From the instructor's perspective, the instruction is clear and unambiguous. From the instructed person's perspective, the instruction may also appear unambiguous but that is because they believe their interpretation of the instruction is correct. It is not until they perform the task incorrectly that the error of their interpretation is revealed along with the ambiguity of the instruction.

Take the example of sharpening a pencil; the person performing the task may use a child's pencil sharpener, an electric office pencil sharpener, a knife, a razor blade, a scalpel or even a grinding wheel. Any of these tools could be used to sharpen the pencil but there is no mention of any of them in the instruction/label, "sharpen the pencil". We could clarify the instruction by including the tool, e.g., sharpen the pencil with a scalpel, but there still exists a risk of misinterpretation. Unless the person performing the task has had experience in using a scalpel to sharpen a pencil they could conceivably sharpen it with the blade pointing towards them, slip and cause themselves a serious injury. The instruction could be further refined to, "sharpen the pencil with a scalpel making sure that the blade is pointed away from your body so you do not cut yourself". Note the inclusion of the reason for pointing the blade away from the body. Inclusion of the reason for doing something improves acceptance of the task or raises questions about the validity of performing the task. Recording why a task is performed provides a check point for process adaptability in dynamic environments. The "why", in this case, is a warning which could be represented as a selectable icon on the task. Clicking on the icon would display the warning.

In a working environment managers do not have the luxury of providing detailed instruction for every task they assign. Task assignment is typically reduced to the most efficient form of communication, in this case, sharpen the pencil. Mistakes are left to the employee to navigate, and employee experience is expected. This highlights a very important point, what is the purpose of process models? If process models are to be used as a form of instruction, then are they to be used as a method of efficiently issuing commands, or are they to be used as a repository of knowledge that ensures consistent quality in performance? If the issuing of commands was relegated to process models displayed on devices then employees would have immediate instruction and managers would have more time to perform other duties. However, it is conceivable that mistakes could increase due to the absence of non-verbal communication between manager and employee. This emphasises the importance of detailed instruction if process models are to be used in this manner. An employee must be provided with adequate warnings and detailed instructions about how to perform a task if compensation claims are to be minimised.

4.8 The Purpose of a Process Model

The different uses of process models mentioned in chapter 1 and reiterated here produce subtly different process models that deserve further investigation. In most cases process models provide a managerial perspective of workflow within an organisation. They communicate the flow of activities and the movement patterns of employees in the workplace. Process models can be optimised for efficiency and quickly checked for compliance. From a business analyst perspective this is by far the most common use of a business process model. A manager, on the other hand, will consult a business process model and issue instructions to employees. An IT specialist may translate a business process model into execution code such as WSBPEL [14] to coordinate web services. An employee could conceivably use a business process model as an instruction manual to explain the how-to of a particular job.

The study revealed that process models developed for the purpose of process improvement represented actions at a much higher level of abstraction than process models developed for process execution. Process improvement has an administrative agenda whereas process execution has an instructional agenda. The first approach focuses on what is to be done while the second approach focuses on how it is to be done. An administrative process model, within this context, is defined as one used to provide information about a process performed by others. An instructional process model is thus defined as one used to provide direct instruction to a person performing a task.

There is a clear distinction between process models used for instruction and those used for administrative purposes. The administrative process model is typically used for analysis and efficiency improvements. The model is interpreted by a manager who in turn instructs employees to perform the tasks. The model was never designed to be a set of instructions to be read by an employee. This underpins a fundamental challenge in any organisation, when an employee is given a task to perform, how much instructional information is required to ensure the task is performed to a consistent standard?

Detailed instruction is often deemed unnecessary for a skilled employee because they understand the detail involved in performing the more abstractly defined task. It is important to note the use of the word "understand" as it refers to the relationship between levels of detail in process models. A task in one process can often be broken down into a sub-process containing more detailed instruction sets. In this regard the instruction set contained in the sub-process under-stands or stands under the original parent process. This parent/child relationship between processes is referred to as "levels of abstraction" or "granularity" and can often lead to miscommunication in the work place when unskilled employees misinterpret an abstract level instruction given by a manager. This generalisation-specialisation relationship between different levels of abstraction in process models is formally defined in [88].

Sub-processing provides a mechanism by which motivationally different process models can be interconnected. As a generalisation it could be said that instructional process models can be represented as sub-processes of administrative process models. However, what one person considers an administrative process model could be considered an instructional process model by another person. In theory there appears to be no reason for concern about how the purpose of a process model will affect its development. In practice, though, the purpose of the process model was found to play a significant role in its development. The two different purposes for which process models are utilised, administrative and instructional, were often reflected in the phraseology of task descriptions.

4.9 Sentence Structure of Task Labels

The type of sentence used as a task label in an administrative process model is typically declarative, written from an observational perspective. Instructional process models are used to issue commands so an imperative sentence is required to represent a task. To illustrate this let's take a simple example of a pharmacy technician preparing a chemotherapy drug (see Fig:4.3). In this study the process analyst, after consulting with the technician, initially labelled the task, "Chemotherapy drug is prepared" and included it in a separate swimlane representing the technician.

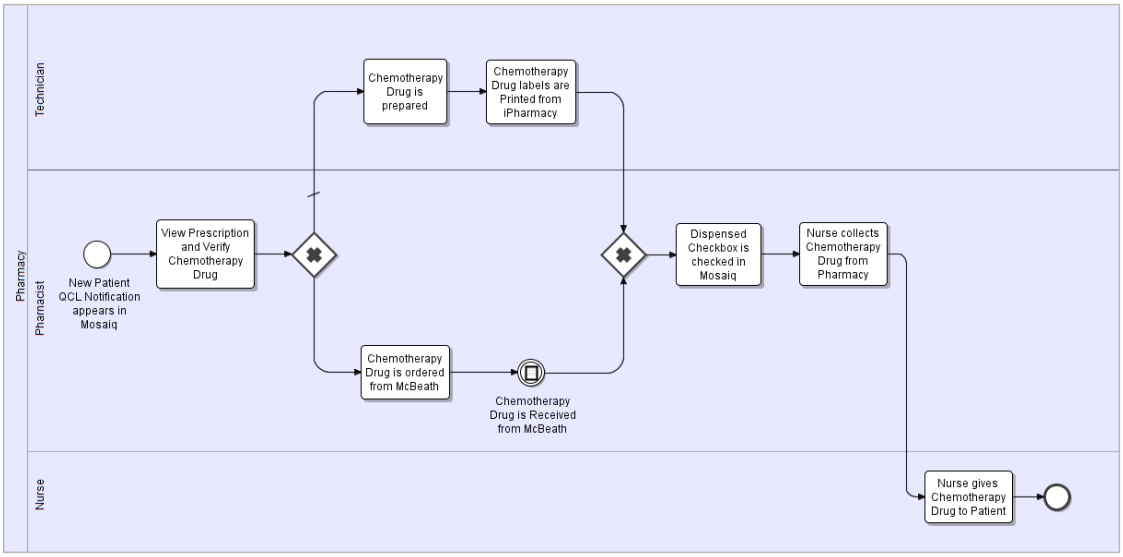

Figure 4.3: An administrative BPMN model for dispensing drugs in a hospital pharmacy.

When shown the finished process model, the technician and all related staff agreed that the process model accurately represented the actual process. However, if this process model was to be used for instructional purposes, the task, "Chemotherapy drug is prepared" could indicate to the technician that the drug had already been prepared. This is because the task label is a declarative sentence that describes the action rather than an imperative sentence that issues a command.

Hospital staff all agreed that the process model was an accurate representation because they were viewing it from an observational perspective. The declarative sentence agreed with that perspective. If we want to use this task in an instructional process model (see Fig:4.4) then we would use the imperative version of this task label, "Prepare chemotherapy drug". This task label makes it clear to the technician what they must do. The ambiguity of this command is acknowledged but it does illustrate how the purpose of the process model affects the types of sentences used in its development. The ambiguity is the result of process abstraction and can be easily resolved using sub-processing techniques where the actual drug required would be first established before preparation.

The revised version of this process model, using imperative sentences for task labels, was shown to all hospital staff involved in the process and every one asked, "what had been changed?" The process analyst found that the changes had no significant effect on their ability to analyse and optimise the model. An obvious conclusion is therefore that an instructional process model can be used for both giving instruction and process analysis. However, the same cannot be said for an administrative process model.

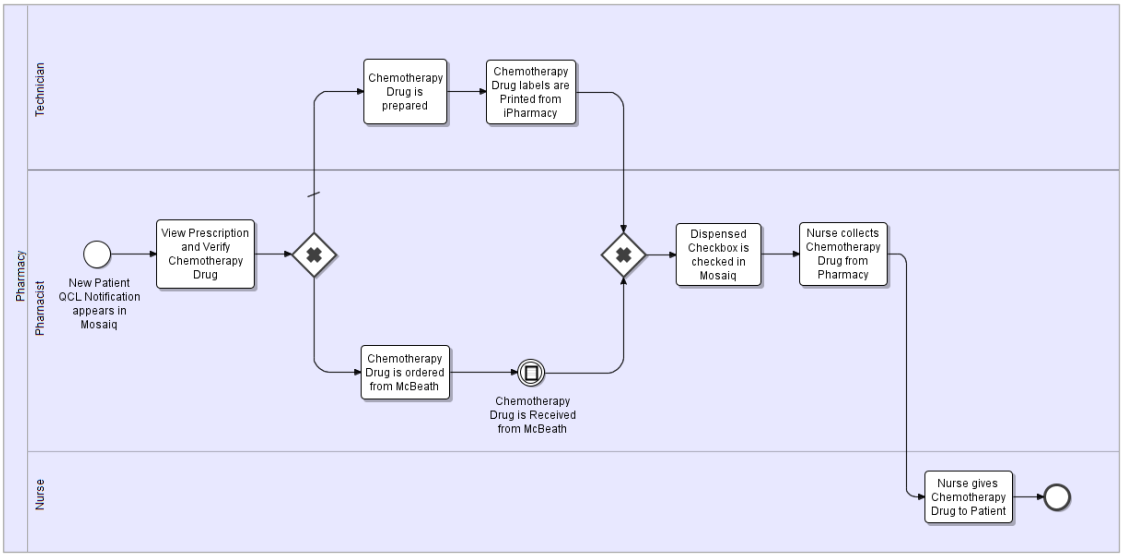

Figure 4.4: An instructional BPMN model for dispensing drugs in a hospital pharmacy.

No examples have been found of BPMN models being used in an instructional capacity so it is understandable that no attention has been given to this inconsistency in task descriptions. It is therefore not surprising to find these same variations in task descriptions occurring within the BPMN Specification. In Figure 10.125 of the BPMN 2.0.2 Specification (see Fig:4.5) the common inconsistency in activity labelling practices can be seen. The following is a list of sentences from the model used to describe tasks. They have been labelled as being either imperative or declarative sentences.

- "Accumulate Requirements" – Imperative
- "Develop Product" – Imperative
- "Sell to Customer" – Imperative
- "Verify Requirements" – Imperative
- "Consulting Required" – Declarative
- "Bugs Diagnosed" – Declarative
- "Develop Patch" – Imperative

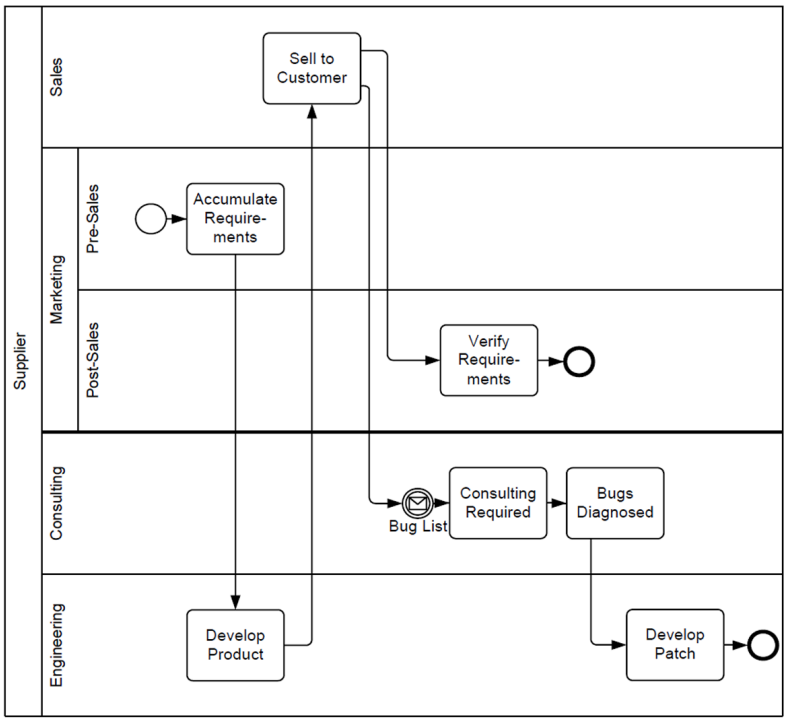

Figure 10.125 – An Example of Nested Lanes

Figure 4.5: A process model example from the BPMN 2.0.2 Specification [111]

Further inconsistency is demonstrated in Figure 11.4 in the BPMN 2.0.2 (see Fig:4.6). The six choreography tasks appearing before the end event in this process use the following types of sentences as labels:

- "Update PO and Delivery Schedule" – Imperative
- "PO and Delivery Schedule Mods" – Declarative
- "Confirmation of Delivery Schedule" – Declarative
- "Retailer Confirmation Received" – Declarative
- "Accept PO and Delivery Schedule" – Imperative
- "Finalised PO and Delivery Schedule" – Declarative

In all fairness, the BPMN 2.0.2 Specification primarily uses imperative statements as labels in the examples provided but there is no specific rule that states, all activity labels must be written as imperative sentences. A process model containing declarative activity labels has limited utility whereas models containing only imperative activity labels can be utilised for both analysis and instruction.

Using imperative sentences to provide instruction is only the first step toward instructional process models. The imperative sentence must be unambiguous if it is to be effective. This became evident in the aerospace industry when writing

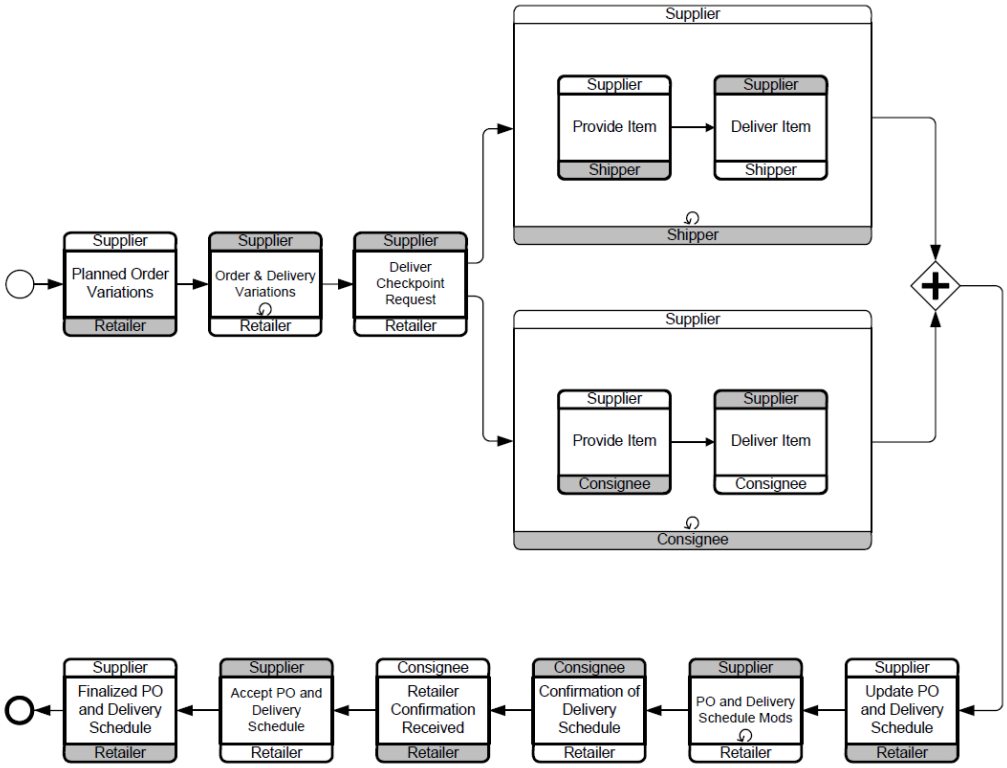

Figure 11.4 – The corresponding Choreography diagram logistics example

Figure 4.6: A Choreography Diagram example from the BPMN 2.0.2 Specification [111]

maintenance manuals. To minimise the ambiguity of maintenance instructions a controlled natural language (CNL) was developed by the Aerospace and Defence Industries Association of Europe (ASD) called ASD Simplified Technical English (ASD-STE100) [50]. This CNL prioritises the disambiguation of human interpreted text because it is designed to communicate instructions to humans. Other CNLs focus on the disambiguation of instructions from humans to machines. Still other CNLs like Attempto [43, 46, 44] strive to bridge the gap between human readability and machine interpretation. Writing in a CNL requires specialist training but most can be read without any training. In fact, someone without any CNL training would not recognise whether a sentence had been written in CNL.

4.10 Representing Urgency

During the study clinicians expressed a concern about BPMN's ability to express urgency. The concern was about whether an employee had to follow the instructions of a process model to completion before undertaking any other task. Clearly this would be impractical in a clinical environment. Life and death situations cannot be put on hold until someone finishes what they are doing. If we think of a process mo-

del as an instruction manual, in the clinical situation clinicians understand a process as an ordered list of patients moving from presentation to completion of treatment. After entering the medical situation, patients undergo a triage process to obtain a clinician assessment, and then, undergo a treatment process where they are assessed for treatment, which results in prioritised appropriate therapy or therapies designed to achieve the outcome desired within the treatment process (e.g., cure, palliation). The prioritisation is the method by which the component therapy tasks are coordinated and executed to complete the treatment process. Each of these therapy tasks has its own internal process, on which the treatment prioritisation might impose a change on the therapy prioritisation. Typically BPMN would represent therapy tasks as sub-processes of treatment processes so the ordered sequence of therapy tasks will impose prioritisation constraints on the component tasks within them.

As a result, task priority does reflect urgency, and this will usually be applied from the initial assessment, although a priority may change from day to day or minute to minute given a patient's condition. For example, if an urgent case deteriorates and becomes futile, the priority is immediately updated; or if a routine case is found to be deteriorating, its priority may be elevated and furthermore therapies may be reordered or rejected. Each therapy task in the treatment process is refined into therapy sub-processes which act as instructions for the care provider. A care provider may be juggling many of these processes at the same time. At this level, urgency is highly dynamic and it is impractical to consider modelling it. Care providers need to be notified of impending urgency and tasks reallocated when a care provider is unable to complete the necessary task. If an automated management system is to determine if a task is being performed within the allocated time frame then it will require interaction with the care provider. The system must be able to ask the care provider whether they have started the task and the care provider must be able to notify the system that they have started the task. A care provider must be able to cancel a task at which point the system will endeavour to locate another care provider that can fulfil the task. The system should be aware of critical time frames and be able to adjust for delays.

The concern about whether a BPMN model can adequately represent urgency stems from a fundamental difference between design-time and run-time process models. A design-time process model is a generalisation of a run-time instance. At design time we can experiment with different configurations to optimise performance but there will always be extenuating circumstances that occur at run-time that will force deviation from the original plan, i.e., the design-time process model. At run-time the process model is utilised as an instruction set to a workflow management system (WfMS). However, the WfMS needs to respond to dynamic changes in the environment. To achieve this, a WfMS needs to assemble a process model on the fly

or compensate for deviant behaviour. To do that it needs to be able to assess the situation and choose appropriate actions to achieve a desired goal. This involves AI planning of the type discussed in chapter 2.12. It is therefore clear that the concern about whether BPMN can model urgency is more of a question of whether a WfMS can respond to urgency. Static process models can only offer a prescribed course of action leaving the WfMS incapable of compensating for deviant behaviour by the user. The approach in [107] allows a WfMS to compensate for deviant behaviour based on activities but does not offer an ability to evaluate the environment. A WfMS that can work with semantically annotated process models can intelligently respond to dynamic situations.

4.11 Future Vision

Workflow Management Systems (WfMS) in health care is not a new idea. Many languages [42, 136, 28, 22, 107] have been developed in a move towards dynamic response WfMSs. In 2001, Dwivedi et al. [36] argued that WfMS, mobile and internet technology would all become standard practice in the future of health care. Already there are mobile apps assisting clinicians in the field [148] but this vision is slightly more ambitious.

With established instructional process models it will be possible to integrate them into a distributed management system where mobile devices can be used to alert and coordinate clinical staff during emergency situations. Imagine a hospital working environment where health care workers are coordinated by computers. Each health care worker carries with them a mobile device that presents tasks that need to be completed. The tasks are automatically prioritised and specifically target the health care workers qualified to perform them. A health care worker commits to a task by selecting it on their mobile device then follows the instructions provided. They can check the context of the task and access more detailed instructions about how the task is to be performed. The mobile device directs them to the location where the task is to be performed and prioritises requests based on the proximity of the health care worker to the task execution area. Patient details can be entered into the mobile device which can plan an appropriate course of action.

Such a physical device is well within the capabilities of our current level of technology. However, the information required for such a device to perform effectively does not yet exist. Semantic effect annotation of process models provides a framework for collecting human experience and relating it to actions performed in the workplace. That experience can then be utilised by machines to suggest appropriate actions in response to given situations.

Consider the following example. A nurse, working on a hospital ward, has just taken the blood pressure of a patient and records the reading in a mobile device hypothetically referred to as a Pocket Health Care Assistant (PHA). If the blood pressure reading is too high or too low then the system would notify a doctor within the immediate area otherwise the PHA recognises that the task has been completed and presents the nurse with a list of tasks currently on offer in the working environment. By selecting a task, the nurse commits to performing it. The displayed tasks have been automatically prioritised and targeted specifically to health care workers based on their skills and proximity to the location where the task is to be performed. The PHA can be used to check the identity and medical records of a patient and alert the appropriate clinicians of abnormal test results.

There may be other health care workers equally qualified within close proximity and any one of them can accept that task. Once the task is accepted, it no longer appears on any other PHAs. However, the health care worker can decline an accepted task if another emergency occurs which would automatically reinstate the task on all PHAs of health care workers suitably qualified.

The health care worker can view all information related to the task to understand the context in which the task is to be performed or perhaps gain a more detailed set of instructions for further clarification. Coordination is managed by a networked back end WfMS whereas communication is the principal function of the PHA so it is vitally important that all forms of instruction are presented unambiguously. This may require multiple delivery strategies, e.g. text, diagrams, video and contextual dictionaries.

Although fictional and not as sophisticated as a Star Trek^{TM}Tricorder [90], the PHA concept provides a valuable research blueprint for health care improvement. If such a device is to be created, it will require a framework for recording, storing, analysing and delivering instructional information. However, let's not get carried away with the idea that a PHA can solve communication problems within the health care industry. In a realistic working environment there may not be enough time for health care workers to review all the clarification material available on the PHA and thus they will be forced to rely on experience to interpret the immediate data presented. Life and death situations require urgent responses, which means the instructions presented on a PHA must be succinct and unambiguous.

4.12 Summary

This chapter discusses how the ProcessSEER framework can be utilised for detecting *treatment conflicts* between co-incident treatment protocols. Also covered in this chapter is the potential for the framework to support a dynamically reactive Work-

flow Management System and the criteria required for that implementation. One of the principal criteria being task labels written as imperative sentences in a controlled natural language. The case study revealed important insights into human-computer interaction problems with the ProcessSEER tool that have directly influenced its development. The Resource Description Framework was adopted as a standard *Effect* input method for the ProcessSEER tool as a direct consequence of this case study. A concern expressed by clinicians about representing urgency in a BPMN model has also been addressed. Finally a future vision has been provided about how the ProcessSEER framework can support hand-held medical assistant devices offering real-time evaluation of patient conditions with appropriate decision support.

Chapter 5

CrowdSourcing Annotations

The following general principles have been employed in the development of a framework for supporting crowdsourcing of process effects:

1. Inconsistencies serve as triggers for discussion, negotiation, deliberation and reconsideration.

2. Inconsistencies can be detected (or surfaced) either directly (e.g., when explicitly specified effects contradict each other) or indirectly. The indirect manifestation of effects is supported by computing the *process-specific consequences* of a set of effects.

For instance, a set of immediate effects of a task might mutually be consistent, but lead to inconsistencies that manifest downstream in a process when these effects are accumulated with downstream effects, or when these effects do not lead to the expected effects at some downstream milestone. The following techniques are offered for supporting crowdsouring:

1. Belief merging techniques can be used to detect and resolve explicit inconsistencies.

2. The proportion of a stakeholders effect specification that are retained in the final set of effects can serve as a "credibility indicator" of the kind that is used in a variety of recently proposed crowdsourcing frameworks.

3. Techniques for surfacing indirect inconsistencies by using effect propagation (two immediate effects for the same task, e1 and e2, are elicited from 2 stakeholders - these turn out to be mutually consistent and are combined into a single immediate effect - but these lead to effect scenarios at a downstream milestone which do not correspond to the expected effect scenarios). Here the inconsistency is essentially between a stakeholder's understanding of the effects of a task, and the same stakeholder's understanding of how these effects

play out in the context of a process design, or between stakeholder1's effects and stakeholder2's expectations.

Methods for optimising the quality and quantity of process model effect annotation acquired through consultation with domain experts have been previously described. Those techniques can now be applied to acquiring effect information from multiple domain experts. Recording effects, using a standardised domain ontology, is effectively building a multi-perspective knowledge base from which outcomes can be derived that may have otherwise remained hidden. A multi-perspective view also includes observations from different levels of abstraction, e.g., an operations manager will most likely give a more abstract account of the effects of a process than those given by a trades person. Sub-processes in process models provide an ideal means for recording and representing these different levels of abstraction. The effects described within a sub-process give a more detailed account than the effects associated with its parent process.

Crowdsourcing is often closely associated with social media. In this regard it is primarily concerned with mining public opinion. Opinion is an easy thing to offer and requires no qualification whereas knowledge needs to be validated. For this reason the scientific community has a peer review system and Wikipedia is policed by other members of the contributing public. While these methods may not guarantee one hundred percent accuracy they minimise the introduction of erroneous knowledge.

Crowdsourcing within the context of semantic effect annotation is concerned with the assimilation of knowledge rather than opinion and thus requires validation. Semantic effects are elicited from experts, i.e., people who are trained to perform the activities under review. Crowdsourcing of semantic effects must therefore be confined to the contributions from experts with the appropriate skill sets. This occurs in the process model repositories of organisations. Although somewhat limited when compared to the data generated by social media, the same crowdsourcing objectives can be applied to this limited data set. The goal of crowdsourcing process models is to combine models of similar processes into the most efficient and comprehensive model for all organisations involved.

Consider two separate hardware retailing outlets belonging to the same parent company, Store A and Store B. They both use the same process for packaging and shipping goods to their customers (see Fig.5.1). However, Store A has calculated its price per item based on expected loss or damage during transportation whereas Store B has factored an insurance cost into its pricing structure. Although the business process models look identical, when the effects are compared it reveals a different story. Both process models contain a task labelled "Check if extra insurance is necessary".

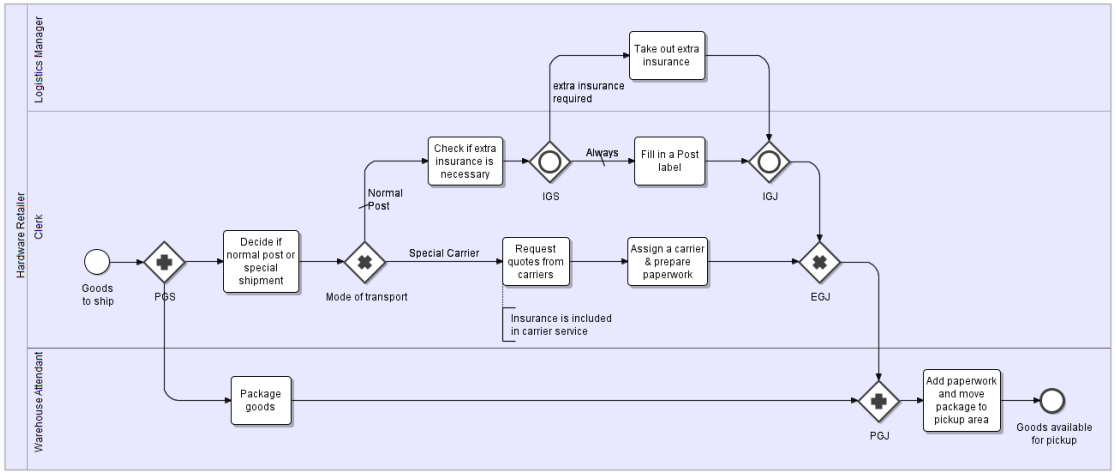

Figure 5.1: A warehousing process model.

The following two immediate effect scenarios, written as SPOTON tuples (see 6.1), have been annotated to this task in the process model of Store A.

Store A EffectScenario1($Aes1$):

$Aes1_1$: $(goods, isFragile, true, a, b, -)$

$Aes1_2$: $(transportationInsuranceForm, isPartOf, shippingPaperwork, r, o, -)$

$Aes1_3$: $(insuranceRequest, isPrintedOn, transportationInsuranceForm, r, o, -)$

Store A EffectScenario2($Aes2$):

$Aes2_1$: $(goods, isFragile, true, a, b, +)$

$Aes2_2$: $(transportationInsuranceForm, isPartOf, shippingPaperwork, r, o, +)$

$Aes2_3$: $(insuranceRequest, isPrintedOn, transportationInsuranceForm, r, o, +)$

$Aes2_4$: $(transportationInsuranceForm, isInPossessionOf, logisticsManager, r, o, +)$

In $Aes1_1$, 'goods' is the subject, 'isFragile' is the predicate, 'true' is the object, 'a' represents that the effect is an attribute description, 'b' represents that the object is a value of type Boolean and the '-' symbol indicates the negation of this effect. It can be interpreted as the goods are not fragile. EffectScenario1 explicitly states that no insurance request is generated, indicated by the negation symbol '-' at the end of each of the remaining Effects. Since there is no transportation insurance form there is no need to track who is or is not in possession of it. Consequently there is no effect that references this.

In $Aes2_1$, the '+' symbol at the end of the tuple indicates that this effect is a positive assertion meaning the goods are fragile. Consequently an insurance request is generated and the person in possession of the transportation insurance form is recorded.

Store A has been in operation for considerably longer than Store B and has therefore collected significant data to support its projected damage bills. Based on this data Store A has negotiated a lucrative agreement in which the logistics

company covers all damages of non-fragile goods. That means that Store A can offer transportation costs marginally less than Store B.

Now let's look at the same task modelled in the Store B's process. The following two immediate effect scenarios, written as SPOTON tuples, have been annotated to this task in the process model of Store B.

Store B EffectScenario1($Bes1$):

$Bes1_1$: $(goods, isFragile, true, a, b, -)$

$Bes1_2$: $(transportationInsuranceForm, isPartOf, shippingPaperwork, r, o, +)$

$Bes1_3$: $(transportationInsuranceForm, isInPossessionOf, clerk, r, o, +)$

$Bes1_4$: $(standardInsuranceRequest, isPrintedOn, transportationInsuranceForm, r, o, +)$

Store B EffectScenario2($Bes2$):

$Bes2_1$: $(goods, isFragile, true, a, b, +)$

$Bes2_2$: $(transportationInsuranceForm, isPartOf, shippingPaperwork, r, o, +)$

$Bes2_3$: $(premiumInsuranceRequest, isPrintedOn, transportationInsuranceForm, r, o, +)$

$Bes2_4$: $(transportationInsuranceForm, isInPossessionOf, logisticsManager, r, o, +)$

There are two scenarios that contain similar effects as those from Store A. However, Store B issues an insurance request form for every item it ships. Semantic effect annotation performed at each location allows these types of discrepancies to be detected whereas the process models alone would indicate that both Stores are following the same procedure.

Semantically annotated business process models capture the unique perspectives of many different contributors, the business analysts, the domain experts and the knowledge engineers. Each process model can contain the views of more than one domain expert. Once a standard process model (i.e. contains no semantic effect annotations) has been designed it is etched in stone so to speak. Its purpose is repeatability and predictability for maintaining quality control. It may be altered to improve efficiency but essentially it represents a rule, "this is how we do this". The activities in the process model have been selected and ordered based on personal observations by a domain expert. Human beings typically filter their sensory input based upon what captures their interest. Two people experiencing the same situation will notice different things about their environment based on what they are interested in or what captures their attention. In a medical setting, what presents as a significant symptom to one doctor may be overlooked by another. The observations of a single individual are limited so by combining individual observations of the same event it is possible to arrive at a more complete description of the situation upon which to decide appropriate actions. The parable of the elephant and the blind men emphasises the need for combining perspectives to arrive at a more complete truth.

Combining different perspectives is inherently difficult because of the limitations of communication. In this sense communication is defined as the transferral of information from one conscious entity to another. If we consider a piece of information being passed from a transmitter to a receiver then no problem exists if the transmitter's encoder uses the same cipher as the receiver's decoder. The information is communicated without loss of semantics. However, communication among humans is flawed because the encoder of one person often does not match the decoder of another person. People can impart their experience (encode) to one another but each person will interpret (decode) the communicated information through their own filter of experiential knowledge thus the meaning of what is said or written can often become distorted by the receiver.

Professionals from different fields of expertise often use the same word to represent different things. Take for example the word, 'ontology', according to the Oxford Dictionary, 'ontology' is "The branch of metaphysics dealing with the nature of being". In this case the word, 'ontology', is a noun denoting an action, i.e., the study of the nature of being. The Oxford Dictionary also gives an alternate meaning commonly held among computer scientists that an ontology is "A set of concepts and categories in a subject area or domain that shows their properties and the relations between them". In this second definition the word, 'ontology', is a noun denoting an object, i.e., a structured set of concepts. A philosopher's interpretation of the word, 'ontology' will therefore be different to the interpretation of a computer scientist. It is ironical that the science of ontology, that seeks to categorise concepts into a standardised semantic representation, is itself semantically contested.

Despite the controversy surrounding the word, 'ontology', the idea of using an ontology to standardise semantics provides a platform for recording multi-perspective experiential knowledge that is semantically consistent. In the case of process models, the experiential knowledge will be limited to an observational perspective, obviously lacking other sensory data that would normally be processed by the human brain. However, observational data underpins many of our greatest scientific discoveries therefore capturing and recording that data in a semantically consistent form allows us to combine these observations and avoid misinterpretation.

As previously stated, communication is one of the biggest challenges when eliciting knowledge from other people. A business analyst will typically not understand the nuances of ontology construction but they will be the conduit through which the experience of an expert is transferred to a knowledge engineer. As such, they run the risk of distorting the knowledge with their own understanding thus creating a world according to the business analyst. For this reason it is imperative that the knowledge engineer engage directly with the expert. However, the knowledge engineer, without context, will typically lack the understanding required to ask the expert

the necessary questions for acquiring the knowledge. Enter the process model. The model provides the context. Knowledge acquisition therefore becomes a three way operation in which the domain expert describes the process to the business analyst, the business analyst develops the process model and annotates the effects then generates a prototypical ontology which the knowledge engineer uses to integrate the terminology into a domain ontology in consultation with the domain expert. It may also be the case that process model discrepancies are revealed when the knowledge engineer consults with the domain expert. This combined approach helps minimise interpretation bias from skewing the data provided by a single individual but an effect scenario still represents the observed state of the world from an individual perspective. When multiple experts perform the same activity and report their individual observations the objective is to expand upon these individually observed states by combining them.

Before combining the effect scenarios of two identical activities it is necessary to consider the following questions:

1. Do both activities originate from process models designed to achieve the same purpose?

2. Do both activities contain the same effect scenarios?

3. Do all effect scenarios remain disjoint or can some be merged?

Question 1 is concerned with how context affects the reported state of the world. As a general rule identical tasks from process models created to achieve different purposes are not considered. For example, a task, "Send Email", could be used in many different process models each providing unique but unrelated contextual information. However, an event is initiated outside the process yet effects changes in the artifacts associated with the process. Combining the effect scenarios of identical events from different process models can reveal important collaborative details discussed later in the section on event merging. Also explored in this section is the possibility of context independent tasks by removing contextual effects from immediate effect scenarios.

Question 2 is concerned with identifying the differences between unique observations. Different effects or different numbers of effect scenarios are triggers for discussion, negotiation, deliberation and reconsideration. Techniques are offered to assist in isolating these differences in preparation for the review process. Effect scenarios may contain erroneous effects, the effects may have been miscommunicated, additional effect scenarios may have been observed or effects may have been described at a different level of abstraction. Understanding the reasons behind the differences will determine which methods are used to combine the effect scenarios.

Question 3 is concerned with how the reported observations are combined. Some effects may be common among all alternate effect scenarios while some effects are disjoint. Some effect scenarios when merged may be consistent but does that mean they should be merged or should they remain separate? Some effect scenarios when merged may be inconsistent but does that mean they should just remain separate? Common effects may need to be copied into disjoint effect scenarios because they are missing. The following section explores techniques for combining the effect scenarios of identical activities.

5.1 Techniques for Combining Effect Scenarios

Semantic effect pair-wise accumulation is a belief revision function [48]. The sequencing of activities in a process model implicitly assigns time indices to the effect scenarios associated with them. This method of time stamping was first proposed by Katsuno and Mendelzon in 1992 [78]. It prioritises the immediate effect scenario (the current state of the world) over the previous cumulative effect scenario (the previous state of the world) because the immediate effect scenario is a more recent account. Individual effects in the immediate effect scenario are considered to be true while effects from the previous cumulative effect scenario are discarded if they cause an inconsistency. The immediate effect scenario always has a higher priority than the previous cumulative effect scenario because it is the most recent observation.

Merging knowledge bases requires some method of prioritisation to determine which assertions hold and which do not. Paolo Liberatore [91] gives examples of prioritisation methods used when merging multiple knowledge bases. However, crowdsourced effect scenarios cannot be prioritised based on time because there is no time line differentiation. When combining multiple accounts of the same activity, there is a need to combine multiple immediate effect scenarios that theoretically occurred at the same time, i.e., immediately following the activity under consideration.

One possible prioritisation method could be based on the experience of the expert providing the account. However, experience measured in time does not always make one expert more knowledgeable than another so this may prove unreliable. Another possible effect prioritisation method could be based on whether the task was actually currently being performed by the expert or whether the expert had simply observed or recalled the performance, the former having more credibility than the latter. The problem with this method is that it cannot prioritise experts with equivalent credibility which will often be the case. While inconsistent accounts may raise questions regarding the credibility of some experts it would be unwise to apply those credibility findings to a prioritisation algorithm. Each case should be

evaluated on its own merits. In this case effect scenario merging is not based on an algorithm but rather an unanimous agreement as to the outcome of a particular activity.

One effect scenario may contain effects not included in the other effect scenario. If the two merged effect scenarios are consistent then they qualify to be merged into a single effect scenario. In some cases this will lead to an expansion of the effect scenario in other cases the two merged effect scenarios may be identical. If the two effect scenarios are inconsistent when merged then this indicates either erroneous effect annotations or that the two effect scenarios identify completely different outcomes witnessed by the different experts. In this case disjoint outcomes are acceptable and no prioritisation of individual effects is necessary. Inconsistent merged effect scenarios act as triggers for discussion among experts, the result being that any errors are corrected and effect scenarios are merged or that the two effect scenarios do in fact identify different possible outcomes. If the number of outcomes increases as a result then the process model may need to be redesigned to accommodate these differences.

There are therefore two options when combining effect scenarios, either merge using arbitration or majority rules. Arbitration aims to retain all effects from all combined effect scenarios which inevitably results in a disjunction of effect scenarios when inconsistencies are encountered. The resulting alternate effect scenarios represent possible worlds that may exist immediately following the execution of the process activity. Merging based on majority prioritises effects that appear in the greatest number of effect scenarios, e.g., if an effect p appears in 2 out of 3 effect scenarios and the 3rd effect scenario contains $\neg p$ then the merged effect scenario will contain p rather than $\neg p$. The majority methods aims to combine all effect scenarios into a single consistent effect scenario that represents the most likely outcome based on the majority of observations.

These methods of merging effect scenarios will determine the final outcome of the merged process models and will ultimately be decided by the experts who provided the original effect annotations. Each semantic effect annotation contains details of the expert who provided it. A business analyst will employ merging techniques to analyse the possible outcomes of effect scenario merging prior to consulting the contributing experts. The analysis data is used as argumentation for deciding which of the merging techniques most accurately represents the observations of all experts.

While there may be the occasion where crowdsourcing produces a large number of models for the same process it is expected that the majority of comparisons will occur between only two models. This section is not concerned with model design although design features may be discussed. Its primary goal is to explore techniques that will assist a business analyst to expand and refine effect knowledge from multiple

sources. The techniques described in this section refer only to combining immediate effect scenarios that belong to identical activities occurring in different models of the same process. The identical activities may contain more than one immediate effect scenario and each activity may contain different numbers of immediate effect scenarios. It then becomes a challenge for the business analyst to identify which immediate effect scenario from one activity corresponds with which immediate effect scenario from the other activity. Before approaching the contributing experts for an adjudication, a business analyst can prepare the following argumentation:

1. The most likely corresponding effect scenarios between two identical activities.
2. The effects from each immediate effect scenario that conflict with the other immediate effect scenario.
3. The effects from each immediate effect scenario that qualify as expansions of the other immediate effect scenario.
4. The possible solutions of merging the immediate effect scenarios of identical activities.

The following sections identify five techniques for comparing and evaluating semantic effect annotations and explore how these techniques can be used to improve process model design. The first technique examines identical activities each containing a single annotated immediate effect scenario. The second technique examines identical activities with more than one immediate effect scenario. The third technique examines substitution and accumulation of immediate effect scenarios to explore indirect inconsistencies. The fourth technique explores the possibility of developing a context independent activity repository and the fifth technique examines how event effects can be used to improve collaboration.

5.1.1 Single Immediate Effect Scenario Merging

When merging immediate effect scenarios of identical tasks it can be expected that the majority of effects will be identical. This helps narrow down any effects that may cause the merged effect scenario to be inconsistent with the background knowledge base. Identifying conflicting effects forms part of the agenda for discussion between contributing experts. Let us consider a case where two organisations are performing the same process. The process models have been designed by two separate business analysts in consultation with domain experts. The domain experts are unique to the organisations. At first glance the process models appear to be identical but these models have been annotated with semantic effects. This allows us to compare the observations that have guided the process designs.

In this example two identical activities from models describing the same process are compared. Each activity has a single immediate effect scenario. Let A and B be two identical activities from two different models describing the same process. Let Aes be the immediate effect scenario of A, i.e., Aes is the set of effects $\{Ae_1, Ae_2, \ldots, Ae_n\}|Ae_i \in Aes$. Let Bes be the immediate effect scenario of B, i.e., Bes is the set of effects $\{Be_1, Be_2, \ldots, Be_n\}|Be_i \in Bes$. Let Mes be the result of merging Aes and Bes. Let KBR be a background knowledge base and set of rules common to Aes, Bes and Mes. It is assumed that $Aes \wedge KBR$ and $Bes \wedge KBR$ are both consistent. Mes is first instantiated as an empty set $\{\emptyset\}$ into which all effects from Aes are copied. Effects from Bes are only added to Mes iff $Be_i \notin Mes$. When every effect from Bes has either been added or discarded then Mes is checked for consistency.

If Mes is consistent then removing the original Aes effects from Mes will reveal additional information introduced by activity B and its corresponding experts. Removing the original Bes effects from Mes will reveal additional information introduced by activity A and its corresponding experts. If no additional effects are detected then Aes and Bes are equivalent thus reinforcing the validity of the observed effects by both experts. It should be noted that consistency is no indication of accuracy therefore the business analyst should never be the adjudicating authority over effect scenario merging. A business analyst upon discovering additional effects through merging will contact the experts involved and alert them to the differences between their observations. This opens a point of discussion between experts and facilitates knowledge expansion. The outcome may be that the additional effects are deemed inaccurate or insignificant to process execution or they may lead to a significant productivity increase or perhaps even a medical breakthrough.

Merging immediate effect scenarios can also result in Mes being inconsistent. One method of identifying conflicting effects involves testing the consistency of Mes for each merged effect. Be_i is identified as conflicting when $Aes \equiv Mes$ and $Mes \wedge Be_i \wedge KBR$ is inconsistent. The operation is repeated for each Be_i merged with Mes until all conflicting effects from Bes are identified. It is equally important to identify the effects from Aes that conflict with Bes. The process is repeated for $Mes \equiv Bes$ each Ae_i is identified as conflicting when $Mes \wedge Ae_i \wedge KBR$ is inconsistent.

It may not be a single effect that is responsible for Mes being inconsistent. Some effects may only cause inconsistency if merged in combination, i.e., $Aes \wedge Be_1 \wedge KBR$ is consistent and $Aes \wedge Be_2 \wedge KBR$ is consistent but $Aes \wedge Be_1 \wedge Be_2 \wedge KBR$ is inconsistent. This is referred to as a co-existent inconsistency. These types of inconsistent effects can be identified using Ginsberg's [54] method of generating maximal consistent subsets from the merging of Aes and Bes. Effects from Aes and Bes not included in the maximal consistent subsets qualify as conflicting. The

maximal consistent subset method can also generate alternate outcomes that can be included in the list of possible solutions.

This type of analysis provides a detailed report of the specific effects in conflict from each expert's unique perspective. It may be that the conflicting effects indicate alternate immediate effect scenarios are required with appropriate decision gateways to process them. The conflicts could also reveal miscommunication within the knowledge acquisition chain or an expert's misinterpretation of events. Whatever the outcome, both organisations benefit from the resolution with improved knowledge about the common process.

5.1.2 Merging Multiple Immediate Effect Scenarios

Activities in processes can generate multiple alternate outcomes in which case the activity in the process model will have more than one annotated immediate effect scenario. This is generally the case with activities that precede decision gateways. It is often the case that alternate immediate effect scenarios will be almost identical, in some cases containing only one effect that differentiates them from each other.

Definition of Alternate Effect Scenarios

Alternate immediate effect scenarios from a single activity when merged will always be inconsistent with the background knowledge base.

Let A be a task annotated with a set of immediate effect scenarios AES such that $AES = \{Aes_1, \ldots, Aes_n\} | Aes_i \in AES$. Let AES_{-i} be the set of immediate effect scenarios remaining in AES after Aes_i is removed. Let $Ae_{i,j}$ be an immediate effect such that $Ae_{i,j} \in Aes_i$.

$\forall Aes_i \exists Ae_{i,j} | Ae_{i,j} \cup Aes_j \cup KBR$ is inconsistent.

Given two identical activities from two different models describing the same process, the immediate effect scenarios annotated to each of the activities are representative of the outcome of the activity as witnessed by different experts. When alternate effect scenarios are present, the consulting expert has reported different possible outcomes generated by the activity. The challenge for the business analyst is to identify which of the alternate effect scenarios from an activity in one model corresponds with which of the alternate effect scenarios from the activity in the other model. Corresponding effect scenarios are identical world state descriptions witnessed from two different perspectives. They may contain different effects or they may be identical. Only the consulting experts will be qualified to declare if two effect scenarios are corresponding. It should never be the role of the business analyst to make this decision. However, the business analyst can perform a preliminary analysis of the alternate immediate effect scenarios from both process activities to identify possible candidates for corresponding effect scenarios.

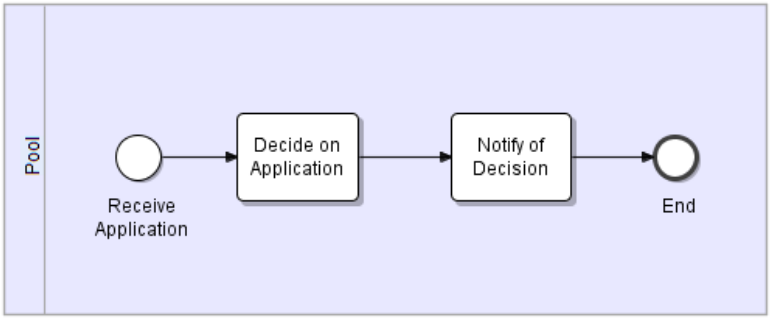

Figure 5.2: Abstract process model.

Two methods are offered for identifying candidates for corresponding effect scenarios. The first method involves comparing immediate effect scenarios with gateway conditions. If the two identical activities from each process model precede a decision gateway then the gateway conditions can be used to match immediate effect scenarios, i.e. immediate effect scenarios from identical activities that satisfy the same conditions qualify as candidates. Standard condition testing is employed. If all the conditions annotated to an outgoing branch of a decision gateway exist in an immediate effect scenario then it satisfies the conditions.

However, some identical activities with alternate effect scenarios do not precede a decision gateway. The process model in Fig.5.2 is an example where alternate effect scenarios are generated by the first task but no decision gateway follows. The process is represented at a high level of abstraction and involves a single actor, the person or group receiving the application and making the decision. Having no decision gateways means there are no conditions against which the effect scenarios can be compared. However, the notification activity implies additional actors that initiate appropriate response actions for each possible decision. This involves a decision gateway even if that gateway is not present within the current process model. It may be possible to utilise the conditions from a gateway in a separate process model that connects with this model but if no such model exists then a second method can be employed.

A Cartesian product style merging technique can be employed to test the consistency of all possible merged pairs. Each effect scenario from one activity is merged with every effect scenario from the other activity and tested for consistency. Consistent merged effect scenarios qualify as candidates for corresponding effect scenarios. This method cannot guarantee the identification of all corresponding immediate effect scenarios but it can reduce the effort required by the business analyst. It provides a starting point for analysis.

The goal of the Cartesian product merging is to identify which immediate effect scenario from one activity corresponds with which immediate effect scenario from

the other activity. In this case each effect scenario from one activity is merged with each of the effect scenarios from the other activity. Let A and B be two identical tasks from different models describing the same process. Each task has two alternate immediate effect scenarios annotated. Let task A be annotated with immediate effect scenarios Aes_1 and Aes_2. Let task B be annotated with immediate effect scenarios Bes_1 and Bes_2. Cartesian product merging involves merging the following pairs:

- $Aes_1 + Bes_1$
- $Aes_1 + Bes_2$
- $Aes_2 + Bes_1$
- $Aes_2 + Bes_2$

There are three possible outcomes for each effect scenario. The outcomes for Aes_1 are:

1. Aes_1 is consistent with only one of the other effect scenarios, e.g. $Aes_1 + Bes_1$ is consistent OR $Aes_1 + Bes_2$ is consistent.
2. Aes_1 is consistent with more than one of the other effect scenarios, e.g. $Aes_1 + Bes_1$ is consistent AND $Aes_1 + Bes_2$ is consistent.
3. Aes_1 is inconsistent with all the other effect scenarios, e.g. $Aes_1 + Bes_1$ is inconsistent AND $Aes_1 + Bes_2$ is inconsistent.

The ideal result is outcome number one which eliminates all other immediate effect scenarios leaving only one. In this case the business analyst can be reasonably assured that the consistent pair are the corresponding effect scenarios from each of the identical activities. Single effect scenario merging techniques can then be employed for further insights.

In the case of outcomes 2 and 3, further analysis is required by the business analyst to determine the corresponding effect scenarios. Additional alternate effect scenarios may be discovered or it may reveal erroneous observations. If an immediate effect scenario is consistent with more than one other immediate effect scenario then it could indicate that it is a subset meaning it may be missing important effect declarations that distinguish it from other effect scenarios. If an immediate effect scenario is inconsistent with every immediate effect scenario of the other activity it could indicate that it is a completely different possible world. These outcomes stimulate further inquiry into the design and expected results of process execution which can only lead to better process design.

5.1.3 Indirect Inconsistency Detection

Consistency checking of merged effect scenarios can reveal valuable insights into process design and expand an organisation's pool of knowledge. Another technique available to the business analyst is to substitute the immediate effect scenarios from one task in an identical model into its corresponding task in the current model then run the accumulation function across the process model. This reveals the impact that different effects have on other tasks within the process. The accumulation function provides immediate feedback about whether the process will execute with the substituted effect annotations. The generated cumulative effect scenarios can be compared to identify changes to the expected outcomes.

The technique is simple. Just copy and paste the immediate effect scenarios from a single activity in one process model into its corresponding activity in the other process model and run the accumulation function. When using substitution it is always better to work backwards through a process model. In this way the changes to accumulated effects are minimised and therefore more manageable.

Substitution will always be used after effect scenario merging but it can also be used prior to merging especially in the case where activities have multiple effect scenarios. Simple substitution of multiple effect scenarios allows the accumulation function to apply conditions on the outgoing edges of downstream decision gateways. This can identify individual effects in the substituted effect scenarios that cause problems which can then be used to identify corresponding effect scenarios. For example, a substituted effect scenario that satisfies the conditions on an outgoing edge of a decision gateway will correspond with the existing effect scenario that satisfies those conditions.

Looking back at the previous multiple effect scenario merging example where task A has two immediate effect scenarios Aes_1 and Aes_2, and task B has two immediate effect scenarios Bes_1 and Bes_2, substitution then accumulation are applied. If a decision gateway immediately follows task A with conditions C_1 and C_2 on the respective outgoing edges the accumulation reveals that Bes_1 is propagated along the outgoing edge containing C_1 and Bes_2 is propagated along the outgoing edge containing Bes_2. Before substitution Aes_1 was associated with C_1 and Aes_2 with C_2. The substitution and accumulation allows us to conclude that Aes_1 and Bes_1 are corresponding effect scenarios and should be merged, likewise for Aes_2 and Bes_2.

5.1.4 Context Identification

Further insights can be gained from comparing activities annotated in context with the same activities annotated in isolation, i.e., the activity is presented to the expert without any indication of the process in which it is involved. Effect annotation is a

process whose first step is to identify artifacts that will be affected by the process under consideration. The effects upon those artifacts by individual activities are then annotated to those activities. Without any contextual information from a complete process model an expert must identify the artifacts that will be affected by the isolated activity then think about the pre and post conditions of those artifacts. Activities considered in isolation therefore elicit both before and after effect scenarios because effects are concerned with change.

The information acquired from a single domain expert may still contain erroneous data or fail to include important data. These types of discrepancies can often be detected by comparing an annotated process activity, which was presented to the domain expert in isolation, to an identical process activity presented within the context of a process model. Presented in isolation, a process task may be insufficient for a domain expert to identify all the important effects. It may equally focus the domain expert's attention on what may otherwise be overlooked in a contextual situation. Analysis of the task in a contextual environment can similarly reveal or conceal important effects. The business analyst needs to allow up to a week before requestioning the domain expert about either the task in isolation or the task in context. Alternatively, if there are multiple processes with which the domain expert is familiar then interviews can be conducted about different tasks over successive days. This helps minimise the regurgitation of memorised effects from previous interviews.

Using this approach, the potential exists for identifying context independent activities, i.e., activities that have minimal immediate effect annotations that can be dropped into any process model. Additional context dependent effects could then be added to the base immediate effect scenarios. A repository of such activities would improve efficiency in process design and effect elicitation.

5.1.5 Event Merging

Events are of particular importance because they are process independent. An event is initiated outside the process but effects change in the artifacts associated with the process. What is interesting about events is that the same event can occur in many different process models but its effects are isolated to the artifacts identified in the particular process model in which it occurs. If identical events from different processes are compared, previously ignored artifacts may be identified. These additional artifacts, when introduced into a process design, can reveal opportunities for collaborative support. An event may impact a number of different departments within an organisation. Knowing the effects that the event has on the processes in each department can assist in the development of more collaborative process designs.

The same effect scenario merging techniques can also be applied to the immediate effect scenarios of events. Contradictory effects require examination and resolution while additional effects identify opportunities for collaboration. Accumulating these merged effect scenarios can identify where additional activities can be introduced to assist other processes within the organisation.

5.2 Summary

This chapter looks at some of the problems encountered with crowdsourced information and offer techniques for combining conflicting effect scenarios. Also described is how such conflicting information can be of benefit to an organisation by revealing inconsistencies. Context identification is shown to contribute to a richer understanding about the effects of activities while event merging can contribute to more collaborative exchange between internal departments.

Chapter 6

Implementation

This chapter describes how the conceptual components from chapter 3, the algorithms from appendix A and the practical application insights from chapter 4 have been implemented. The implemented tool has been named after the framework, "ProcessSEER" which stands for "Process Semantic Effect Evaluation and Reasoning". The "SEER" acronym was considered applicable because the tool provides an insight into future scenarios based on current situations. The ProcessSEER tool is implemented using the Eclipse environment [41], as a plug-in to the STP BPMN modelling tool [37]. The modelling tool provides a graphical representation of process models that comply with the BPMN 1.1 specification [112], i.e, they have an underlying XML format that is platform independent. The BPMN 1.1 specification was current at the time when the ProcessSEER tool was first developed. Not all BPMN elements are recognised by the ProcessSEER plug-in but it does cover the most commonly used. Other BPMN modelling tools, that are more up-to-date with the current standards, are currently under review.

The original version of ProcessSEER provided the user with a single data entry field for annotating *Effect* sentences using the ACE-CNL controlled natural language toolkit [132]. The variety of process models that could be accumulated was extremely limited and plotting *Scenario Labels* was mandatory before accumulation could occur. Accumulated *Effect Scenarios* were output manually and fed into a Prover9 theorem prover [100] for consistency checking. The primary role of this alpha version software was to demonstrate the capability of reasoning with semantic effect annotations.

The tool is still in an alpha stage of development, currently at version 4.0 with development about to commence on version 5.0. The graphic user interface has been significantly altered to accommodate the insights gained from clinical trials (see 4.4). *Effect* annotations are now constrained to binary predicate input using three distinct fields that do not require knowledge of a CNL. The tool now has direct access to a background ontology and reasoning engine, and can compute process-

wide accumulation in real-time. An additional feature has also been included that allows the business analyst to output a rudimentary ontology to be passed on to a knowledge engineer. The rudimentary ontology is used to assimilate any new terms, introduced by the business analyst, into the background ontology. This feature precedes an ability to flag process models as being integrated with the background ontology. An integrated process model will return accurate accumulation results but until such time as the model becomes integrated the business analyst can still continue work with appreciable functionality.

The background ontology has been implemented in the Web Ontology Language or Ontology Web Language (OWL) (see chapter 2.10). OWL is based on the Resource Description Framework (RDF) (see chapter 2.9) and has become an accepted internet standard by the W3C for the Semantic Web (see chapter 2.11). RDF's structure replicates the structure of a sentence by separating its component parts into the Subject , the Predicate and the Object. Constraining input to these three fields has proven to be more effective for data entry than allowing the user the freedom to construct the entire sentence in CNL. The structure maintains an internationally standardised format and focuses the user's attention on the things that are affected by an activity rather than what the effect of the activity is. This simple distinction between how the question is asked has greatly improved the *Effect* data gathered. Using OWL for knowledge representation has further extended the implementation of an *Effect*. Effects have changed from single CNL sentences to what is referred to as a SPOTON.

6.1 A SPOTON State Description

A SPOTON is a six tuple consisting of a **S**ubject, **P**redicate, **O**bject, **T**ype, **O**bjectType and **N**egation (S, P, O, T, O, N). The first three elements S, P, O represent the component parts of a sentence which correspond to an RDF triple. An *Effect*, written in natural language, is a declarative sentence describing the consequent state of a process artifact being the result of some activity. The sentence describing the *Effect* can be broken down into a triple $(Subject, Predicate, Object)$. Using this structure the $Subject$ always becomes any artifact that is affected by an action. The triple can be represented in two ways, as either an Attribute Triple $(artifact, Attribute, Value)$ or a Relationship Triple $(artifact, Relationship, artifact)$. The fourth element, T, refers to the Type of state description and specifies whether the predicate P refers to an Attribute or a Relationship. The Object of a semantic effect statement can be either a Value or another artifact depending on which Type of triple is used. An artifact will always be represented as a string data type whereas a Value could be one of five data types: integer, float, double, boolean or string. The fifth element

in a SPOTON, O, is used to specify the ObjectType so that machines can process the Value correctly. The ObjectType integrates with OWL's Object Properties and Datatype Properties and allows the SPOTON to be stored in an OWL ontology.

When the notion of negated *Effects* is introduced then each of these formats has a different way of expressing negation. A formal binary predicate statement generally takes the form $Predicate(Subject, Object)$. When a binary predicate statement refers to a relationship, e.g., $Inside(Ball, Box)$ then it is intuitive to append the negation to the predicate, i.e. $NotInside(Ball, Box)$. However, when a binary predicate statement refers to an attribute, e.g., $hasPosition(LightSwitch, On)$, it is sometimes intuitive to say, "The light switch is not on", leading to the negation of the Object in the sentence rather than the Predicate,
e.g., $hasPosition(LightSwitch, NotOn)$. The syntax is incorrect for a formal notation and can complicate the task of manual semantic effect annotation for someone who is unfamiliar with such notations. The syntactic placement of the negation symbol should not hamper the construction nor interpretation of a semantic effect annotation. The *Effect* data entry form therefore isolates it from the binary predicate statement to avoid confusion. All semantic effect annotations are positive binary predicate statements. The entire statement is either negated or not by a flag and should not depend on the positioning of that flag within the statement itself. The flag has therefore been included as a separate element N within the SPOTON framework. The acronym SPOTON also refers to the spots (tokens) used in a petri-net. The tokens, generated by transitions, are similar to the *Effects* generated by activities in a BPMN model. *Conditions* can be specified in the same SPOTON format which can then be used to evaluate *Effect Scenarios* much the same as a transition in a petri-net evaluates the number of tokens contained in a place. See chapter 2.2 for more information. In the next section it can be seen how the SPOTON structure influences the user interface design.

6.2 Graphic User Interface (GUI) Design and Data Entry

The ProcessSEER plug-in utilises the STP BPMN modeller interface with a file navigator on the left, the main process model in the centre at the top, a list of BPMN icons on the right and a Property Section at the bottom. The ProcessSEER plug-in generates three Property Section Tabs that appear whenever an appropriate BPMN element is selected in the main process model window. Selecting a task or event will display both the *IEffect* and the *CEffect Tabs* in the *Property Section*. Selecting an outgoing sequence flow from an exclusive or inclusive gateway split or

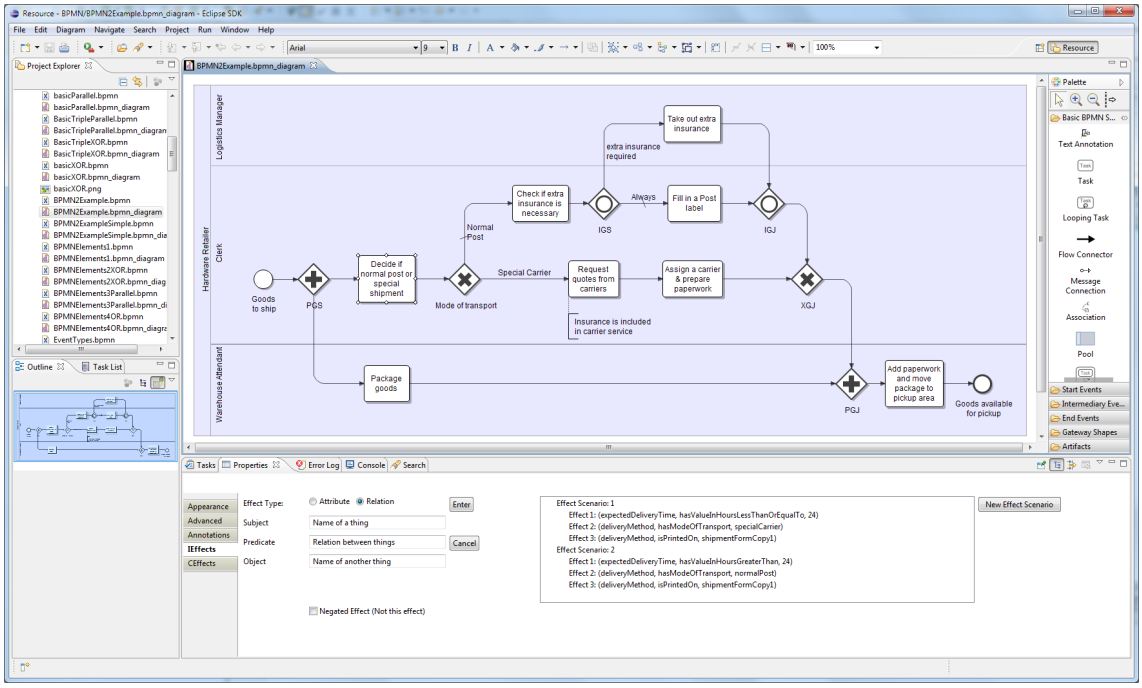

Figure 6.1: The ProcessSEER GUI showing the Immediate Effect data entry tab.

a start event will display the *Conditions Tab* in the *Property Section*. Figure 6.1 shows the ProcessSEER tool GUI with a task in the process model window selected. The *IEffect Tab* is active showing data entry on the left and a display of *Immediate Effect Scenarios* on the right. The *CEffect Tab* is also visible but not active while the *Conditions Tab* is not visible at all because a sequence flow is not selected.

6.2.1 Immediate Effect Tab

New empty *Effect Scenarios* can be added or existing *Effect Scenarios* can be deleted from the right-hand panel in the *IEffect Tab*. These are only *Immediate Effect Scenarios* in this section and each one is a tree display that can be expanded to reveal the *Effects* inside or contracted to show only the *Immediate Effect Scenario*. The panel on the left of the *IEffect Tab* is for entering individual *Effects*. When a business analyst wants to manually annotate an *Immediate Effect* to a BPMN activity they must first either create a new, or select an existing, *Immediate Effect Scenario* in the right-hand panel. Selecting an *Effect* in the right-hand panel will display its contents for editing in the left-hand panel. Figure 6.2 shows the two panels in the *IEffect Tab*.

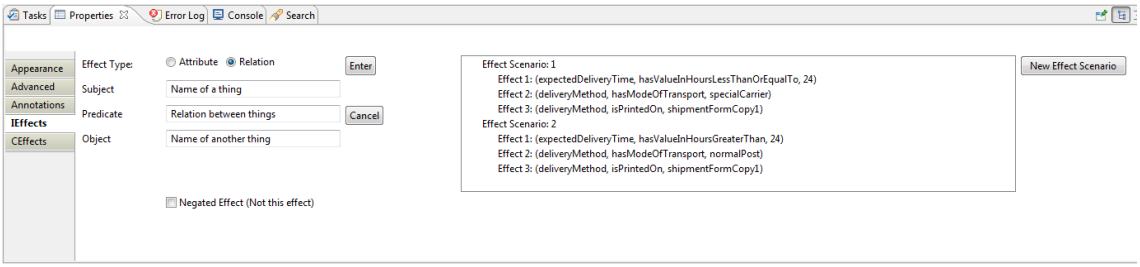

Figure 6.2: The ProcessSEER IEffect Tab showing data entry on the left and Immediate Effect Scenario display on the right.

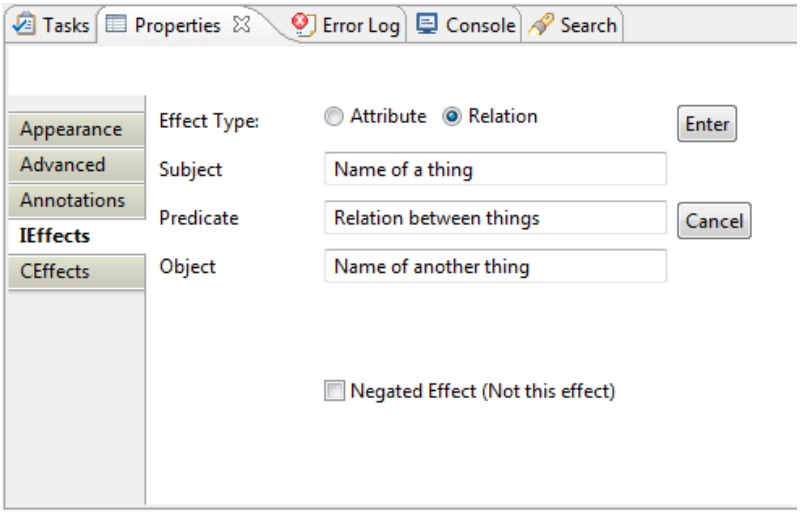

Figure 6.3: Effect data input fields in the ProcessSEER tool.

6.2.2 Immediate Effect Data Entry

Individual *Immediate Effects* are entered in the left-hand panel of the *IEffect Tab*. Figure 6.3 shows the panel consisting of the following data entry elements:

- Three text fields for the Subject, Predicate and Object
- Two radio buttons for the Type of effect statement (Attribute or Relationship)
- A drop down list of ObjectTypes
- A checkbox for Negation.

The data entry panel has been designed around the SPOTON method of representing state descriptions. The business analyst has only to consider the artifact that is being affected and write its name in the *Subject* text field. They then need to consider whether they are describing an Attribute of the artifact or a Relationship between the artifact and another artifact and select the appropriate radio button. In the case of the *Effect* describing an Attribute, the business analyst would enter an appropriate predicate into the *Predicate* text field, e.g., *hasColour* or *hasLengthInCentimetres* (Camel case is used to combine multiple words). These

multi-words become separate entries in an OWL ontology. In the first example, *hasColour*, the *Value* entered into the Object field might be *Blue* in which case *Concept* would be selected from the drop down list as the ObjectType. In the second example, *hasLengthInCentimetres*, the *Value* could be a *float*, an *integer* or a *double*. In the case of an *Effect* describing a Relationship, the business analyst would enter the second artifact into the *Object* text field and write an appropriate relationship predicate, e.g. $isPartOf$, in the *Predicate* text field. Note that the OWL recommended guidelines for naming Predicates are used. Attributes take the form *hasAttribute* and Relationships take the form of $isRelatedTo$.

The business analyst's task of manually annotating *Effects* is significantly simplified. The syntax of a formal notation has been removed and the need to establish the existence of every artifact using CNL is no longer necessary. CNL also allows too much flexibility in the way an effect is written because of its sentence-like structure. A SPOTON constrains the data input yet still maintains a readable format. Formatting of the effect statement into formal notation is performed in the background. The interface does not allow the business analyst to enter *Effects* in a natural language but it does provide a reasonable facsimile.

When combined with an ontology the ProcessSEER tool has the potential to suggest terms to the business analyst, in the data entry fields, and a dialog box provides a convenient method for submitting knowledge for inclusion into an ontology. New *Effects* are like requests to knowledge engineers for inclusion into the ontology. A BPMN model is not fully functional until all the terms used in its *Effect* annotations have been integrated into the background ontology. Once integrated, additional *Cumulative Effects* can be revealed that the business analyst or even the expert may not have considered. Manual *Effect* annotation is a method of knowledge acquisition that contributes to ontology development and provides an immediate return on investment because the models can be immediately utilised. These domain ontologies are conceived as providing significant competitive advantage to an organisation.

6.2.3 Conditions Tab

The *Condition Tab* is almost identical to the *IEffect Tab*. That is because *Conditions* are syntactically identical to *Effects* (see chapter 3.4.2). The right-hand panel displays the *Condition Scenario*. Note that only one *Condition Scenario* can ever be created for each sequence flow. The sequence flow must be an outgoing edge from an exclusive or inclusive gateway split or an outgoing edge from a start event otherwise the *Conditions Tab* will not display. The *Condition Scenario* in the right-hand panel has a tree display like the *IEffect Tab*. *Conditions* can be selected individually and edited or deleted.

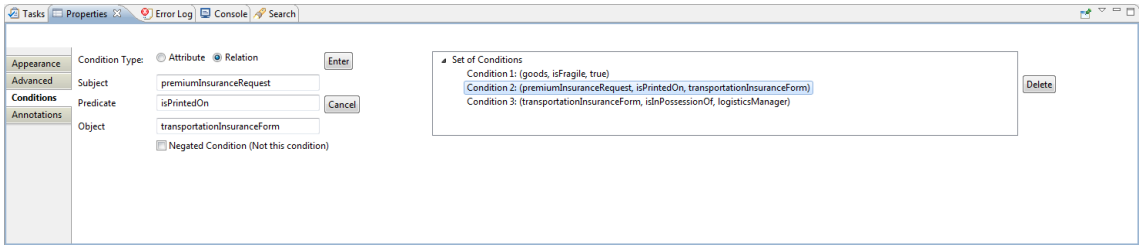

Figure 6.4: The ProcessSEER Conditions Tab showing data entry on the left and Condition Scenario display on the right.

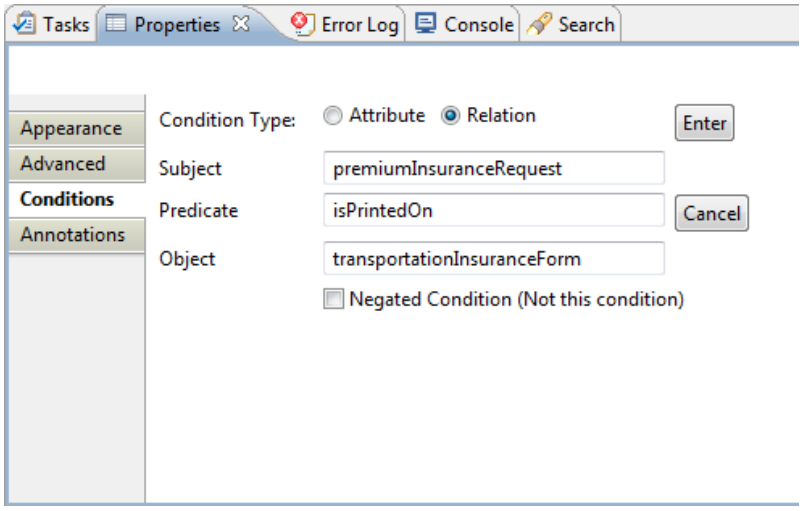

Figure 6.5: Condition data input fields in the ProcessSEER tool.

6.2.4 Condition Data Entry

Condition data entry is identical to *Effect* data entry. The business analyst is describing a state, not one that is in effect but one that should be in effect. A *Condition* must exist in an *Effect Scenario* otherwise the *Effect Scenario* will not be propagated along the annotated sequence flow. All *Conditions* in a *Condition Scenario* must exist in an *Effect Scenario* before it can be passed on to the next activity for accumulation. Once again the artifact under consideration is entered into the Subject field. Whether the *Condition* is describing an Attribute or a Relationship is selected. An appropriate Predicate and Object are entered into the other two fields respectively. If the business analyst wishes to express that the *Condition* should not be the case in any *Effect Scenario* then they will check the Negation checkbox. Entering the *Condition* includes it in the *Condition Scenario*. Figure 6.5 shows the data entry fields of the *Conditions Tab*.

6.2.5 Cumulative Effect Tab

Whenever an activity is selected in a process model the *IEffect* and *CEffect Property Tabs* become visible in the property section at the bottom of the screen. Selecting

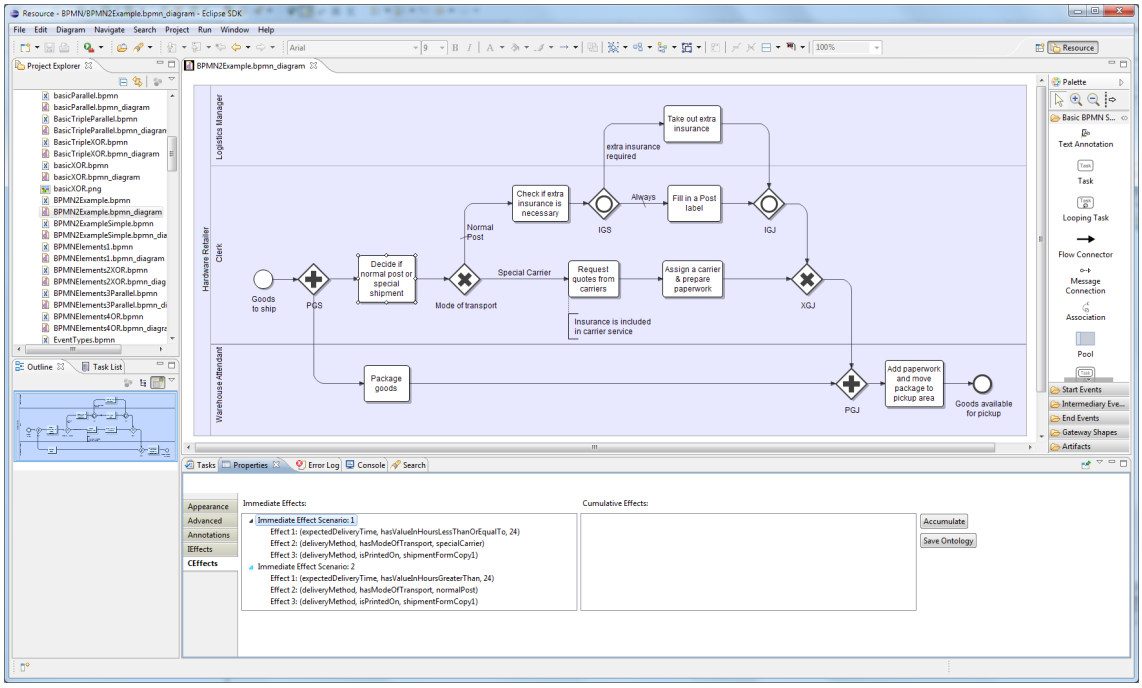

Figure 6.6: The ProcessSEER GUI showing the CEffect Tab for Cumulative Effects.

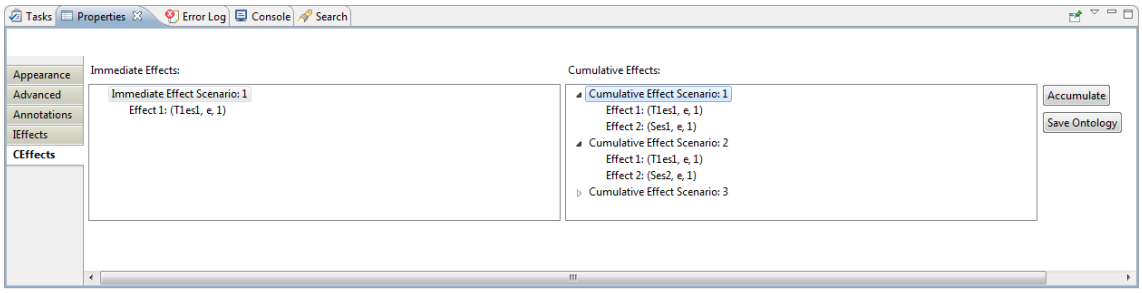

Figure 6.7: Data Displayed in the CEffect Tab. Example taken from a test process model with only dummy data used.

the *CEffect Tab* displays the accumulation interface shown in Fig:6.6. The business analyst can inspect all the *Effects* contained within the *Immediate Effect Scenarios* annotated to the selected activity or the *Cumulative Effect Scenarios* derived from accumulation.

Immediate Effect Scenarios annotated to the selected activity are displayed in the left-hand panel and *Cumulative Effect Scenarios* are displayed in the right-hand panel as shown in Fig:6.7. Each *Effect Scenario* is a tree display allowing the user to expand them to view the *Effects* inside. Clicking on the "Accumulate" button will accumulate all nodes in the model no matter which activity is selected. A progress bar, shown in Fig:6.8, displays which activities are being accumulated.

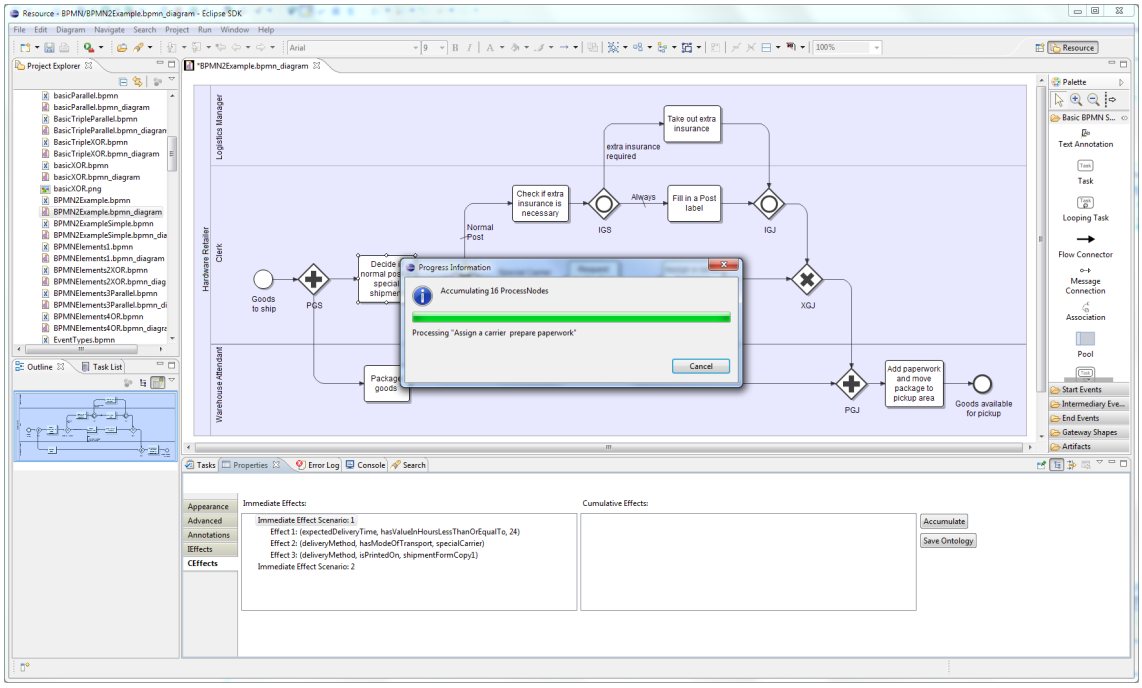

Figure 6.8: The ProcessSEER GUI showing accumulation progress bar.

6.3 Back-end Implementation

This section describes the different classes used to implement the conceptual framework described in chapter 3. The *Effect* class, previously explained, is a SPOTON as is the *Condition* class. These classes contain methods for translating their data into XML format that is stored in the `.bpmn_diagram` file. Each process model in the STP modeller is a combination of two files, a `.bpmn_diagram` file and a related `.bpmn` file. *Immediate Effects* and *Conditions* are stored in the `.bpmn_diagram` file. *Effects* and *Conditions* are two of the three foundational classes used in the ProcessSEER tool. The third foundational class is the *ProcessNode*.

6.3.1 Process Node Class

The ProcessSEER tool is a plug-in to the Eclipse STP BPMN modelling tool. BPMN model elements are generated from an Activity class in the modelling program. The Activity class extends the Eclipse *EModelElement* class which contains an *EAnnotationList* by default. Every BPMN element can therefore accept annotations. A BPMN model has a corresponding XML Metadata Interchange (XMI) file associated with it. *Effect* annotations are represented as text. It can be seen in the following snippet, taken from a ProcessSEER annotated model, how *Immediate Effect Scenarios* (imeffect) are stored in the XMI format. Note there is no *Cumulative Effect Scenario* in this snippet as the cueffect value is empty.

Snippet of a BPMN model in XMI format

```
<vertices xmi:type="bpmn:Activity"
  xmi:id="_57GnQR7hEeGgLZueCUYOzw"
  iD="_57GnQB7hEeGgLZueCUYOzw"
  outgoingEdges="_BFd8MR7iEeGgLZueCUYOzw"
  incomingEdges="_9XAtcR7hEeGgLZueCUYOzw"
  name="Fill in a Post label"
  lanes="_Ig-g8B7hEeGgLZueCUYOzw"
  activityType="Task">
  <eAnnotations
    xmi:type="ecore:EAnnotation"
    xmi:id="_xCJtoMVZEeWcN4gf6mfBqw"
    source="effect">
    <details
      xmi:type="ecore:EStringToStringMapEntry"
      xmi:id="_xCJtocVZEeWcN4gf6mfBqw"
      key="imeffect"
      value="(package,hasTransportationInsurance,true,a,b,-)
             (deliveryAddress,isPrintedOn,postLabel,r,o,+)
             (postLabel,isPartOf,shippingPaperwork,r,o,+)"/>
    <details
      xmi:type="ecore:EStringToStringMapEntry"
      xmi:id="_xCK7wMVZEeWcN4gf6mfBqw"
      key="cueffect"
      value=""/>
  </eAnnotations>
</vertices>
```

Contained within the `<eAnnotations>` tags are three *Effects* in SPOTON format.

```
(package,hasTransportationInsurance,true,a,b,-)
(deliveryAddress,isPrintedOn,postLabel,r,o,+)
(postLabel,isPartOf,shippingPaperwork,r,o,+)
```

In the first *Effect* the Subject is *package*, the Predicate is *hasTransportationInsurance* and the Object is *true*. The three symbol code at the end of each *Effect* refers to the type of *Effect*, the ObjectType and whether or not the *Effect* is negated. The first *Effect* is an *Attribute* with a boolean ObjectType and it is negated. These SPOTON *Effects* are easily translated into OWL assertions or first-order predicate logic. A translation of each *Effect* into first-order predicate logic is provided below.

$\neg hasTransportationInsurance(package, true)$
$isPrintedOn(deliveryAddress, postLabel)$
$isPartOf(postLabel, shippingPaperwork)$

Lists of complex objects like *Effects*, *Effect Scenarios* and *World Lists* cannot be stored in the core modelling program's *EAnnotations*. The ProcessSEER plug-in therefore uses a *ProcessNode* adapter class acting as an interface between the STP Activity class and the ProcessSEER tool. A *ProcessNode* instance stores the complex data types necessary for accumulation and populates the XMI *EAnnotations* with textual representations of the *Effects*.

6.3.2 Effect Scenario Implementation

Effect Scenarios are lists of *Effects* but the implemented class contains a rudimentary OWL ontology (ESO) that is used for reasoning. It is referred to as rudimentary because it may contain terms that are not included in the background ontology. This approach allows a business analyst to use any term they want to describe an *Effect*. New terms are linked directly to the root classes of the ontology having no other links to any other terms. The OWL framework still provides enough flexibility for a rudimentary ontology to be reasoned with as part of the accumulation process. The outcomes may not be absolutely accurate but they are sufficient to allow a business analyst to continue working on the design of a process. During accumulation it is the ESO that is compared with a background ontology to check for consistency.

Adding Effects to an Effect Scenario

Each *Effect* is a SPOTON containing textual representations of the Subject, Predicate and Object of the *Effect* sentence. When an *Effect* is added to an *Effect Scenario* it is stored in both its textual form and its ontological form. In an ontology an *Effect* is an assertion that is constructed from the terms used in the Subject, Predicate and Object of the *Effect*. These terms must also be entered into the ESO. The Subject of an *Effect* will always be an *OWL Named Individual* (an instance of an OWL class). If the *Effect* is of type *Relationship* then the Object of the *Effect* will also be an *OWL Named Individual*. These *OWL Named Individuals* are instantiated in the ESO as members of the root class *OWL Thing*. This is updated once a knowledge engineer integrates these terms into their appropriate classes in the background ontology.

Consider an *Effect*: $isAffixedTo(postLabel, package)$ where $Subject = postLabel$, $Predicate = isAffixedTo$ and $Object = package$. The following examples show how the *Effect* is recorded in an ESO. Declarations and class assertions establish the terms in the ontology.

Example 1: Snippet of an OWL Effect Scenario Ontology showing instantiations of Classes and Individuals

```
<Declaration>
    <NamedIndividual IRI="#postLabel"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#package"/>
</Declaration>
<ClassAssertion>
    <Class abbreviatedIRI=":Thing"/>
    <NamedIndividual IRI="#postLabel"/>
</ClassAssertion>
<ClassAssertion>
    <Class abbreviatedIRI=":Thing"/>
    <NamedIndividual IRI="#package"/>
</ClassAssertion>
```

A package is not a postLabel and this needs to be explicitly stated in the ESO. Individuals are declared as being different things in the following example.

Example 2: Snippet of an OWL Effect Scenario Ontology showing declaration of different Individuals

```
<DifferentIndividuals>
    <NamedIndividual IRI="#package"/>
    <NamedIndividual IRI="#postLabel"/>
</DifferentIndividuals>
```

The Predicate of a relationship *Effect* is also entered into the ESO as an *OWL Object Property* being a member of the root *OWL Top Object Property*. All Predicates newly introduced by a business analyst will be entered into the ESO as *Functional Properties*. A *Functional Property* can only have one value so when the Object of an *Effect* changes during accumulation, an inconsistency will occur between the new state and the previous state. The new state will replace the old. This provides a modicum of reasoning functionality with which the business analyst can perform accumulation on a process model that has not yet been integrated with the background ontology. The business analyst does not have to wait for ontology integration to carry on with their work. Once a new term has been integrated with the background ontology, the tool will use that term rather than create a new one.

Continuing the example reveals the two Predicate entries in the ESO.

Example 3: Snippet of an OWL Effect Scenario Ontology showing an Object Property

```
<Declaration>
    <ObjectProperty IRI="#isAffixedTo"/>
</Declaration>
<FunctionalObjectProperty>
    <ObjectProperty IRI="#isAffixedTo"/>
</FunctionalObjectProperty>
```

Finally with all the terms established in the ESO the *Effect* assertion is entered.

Example 4: Snippet of an OWL Effect Scenario Ontology showing an Object Property Assertion

```
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#isAffixedTo"/>
    <NamedIndividual IRI="#postLabel"/>
    <NamedIndividual IRI="#package"/>
</ObjectPropertyAssertion>
```

If the *Effect* is of type *Attribute* then the Subject is again an *OWL Named Individual* but the Predicate is entered into the ESO as an *OWL Data Property* being a member of the root *OWL Top Data Property*. Once again, this will change when the process model is fully integrated with the background knowledge base. The Object of an *Attribute* type *Effect* can be either a value or another thing. If the latter then the Object is entered as an *OWL Named Individual* otherwise the *ObjectType* data entry field in the ProcessSEER GUI determines the type of Object, i.e., integer, float, double or boolean. An Object value is entered into the ESO as an *OWL Literal*. The Predicate in the following example indicates the weight of the previous package. Note that the Predicate contains an explicit reference to the weight scale (Kilograms). When dealing with measurements it is necessary to include the measurement scale in the Predicate or else it becomes meaningless. When entering the *Effect* $hasWeightInKilograms(package, 2.7)$ we can dispense with the *OWL Named Individual* declaration and *OWL Class Assertion* for package because they already exist. Continuing with the example the following is entered for the *OWL Data Property* and *OWL Data Property Assertion*.

Example 5: Snippet of an OWL Effect Scenario Ontology showing a Data Property Declaration and Assertion

```
<Declaration>
    <DataProperty IRI="#hasWeightInKilograms"/>
</Declaration>
<FunctionalDataProperty>
```

```
        <DataProperty IRI="#hasWeightInKilograms"/>
    </FunctionalDataProperty>
    <DataPropertyAssertion>
        <DataProperty IRI="#hasWeightInKilograms"/>
        <NamedIndividual IRI="#postLabel"/>
        <Literal datatypeIRI="&xsd;double">2.7</Literal>
    </DataPropertyAssertion>
```

Finally one additional feature that was found to be particularly useful was *OWL Annotations*. The details of each expert consulted about a process can be included in the ontology. This only needs to be entered once for the entire process model. If more than one expert has contributed to a model then the business analyst can select the appropriate expert for each *Effect* annotation. When integrating a process model with the background ontology, a knowledge engineer can consult directly with the expert on matters of disambiguation. The following example includes dummy data. Note that the term to which the OWL Annotation applies is encased with `NEW[ ]` to highlight the term as a new entry that must be integrated into the background ontology.

Example 6: Snippet of an OWL Effect Scenario Ontology showing an Annotation Assertion

```
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:label"/>
        <IRI>#postLabel</IRI>
        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
           NEW[postLabel]</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#premium</IRI>
        <Literal xml:lang="en" datatypeIRI="&rdf;PlainLiteral">
           Analyst: Bob Jones
            Analyst Email: bj@organisation.com
            Analyst Phone: 5555 4293
            Expert: Mary Davis
            Expert Email: md@organisation.com
            Expert Phone: 5555 7652
            Date of Entry: 15/9/2013
        </Literal>
    </AnnotationAssertion>
```

The details of the business analyst and experts could easily be linked to an online or local database that would provide automatic updating capabilities thus coordinating the process repository with staff records.

As each *Effect* is added to an *Effect Scenario* a copy of its ESO is populated with the OWL entries shown above and then checked for consistency. If consistent then the *Effect* is added and the copied ESO replaces the existing ESO. If it is inconsistent then *Effects* need to be removed. This occurs regularly during accumulation and a possible worlds algorithm is used to determine the resulting *Cumulative Effect Scenarios*.

Removing Effects from an Effect Scenario

Removing *Effects* from *Effect Scenarios* is not as easy as adding them. The text version of the *Effect* is easy to remove but it is still registered in the ESO. Removing it from the ESO can be very difficult because all the links in the ontology that relate to a particular entry also need to be carefully removed. If the wrong links are removed it will render the ESO useless. The easiest way of removing an *Effect* from an ESO is to first remove the text version of the *Effect* then rebuild the ESO from the remaining *Effects* in the *Effect Scenario*.

6.3.3 Condition Scenario Implementation

Condition Scenarios are lists of *Conditions* but the implemented class also contains a rudimentary OWL ontology (CSO) that is used to check the *Condition Scenario* for consistency. *Condition Scenarios* are only ever used to check for the existence of *Conditions* in *Effect Scenarios*. They only interact with the background ontology when new *Conditions* are added or removed. Adding a new *Condition* to a *Condition Scenario* involves all the same steps as those employed by *Effect Scenarios*. Removal of *Conditions* also employs the same steps as those used to remove *Effects* from *Effect Scenarios*.

6.3.4 KB Manager Class

The ProcessSEER KB Manager class utilises the OWL API to assemble OWL ontologies. It also acts as an interface between the ontologies it creates and an OWL reasoning engine for consistency checking. The tool is currently using the HermiT OWL Reasoner [139] and will eventually include additional reasoners as a user selectable option in future versions of ProcessSEER.

Effect Scenario ontologies are updated during accumulation by passing the SPOTONs to a KB Manager. A new ontology is created from the set of SPOTONs and

returned to an *Effect Scenario* if it is consistent. In the case of adding a new *Immediate Effect* to an *Immediate Effect Scenario*, inconsistency will reject the addition of the new *Effect* (see chapter 3.4.3). In the case of accumulation, an inconsistent *Effect Scenario* is passed to a possible worlds function (see appendix A.3) that generates a set of maximal consistent subsets of the original inconsistent *Effect Scenario*. Each subset becomes a new *Effect Scenario*.

The KB Manager also assembles a rudimentary ontology of the entire process model and outputs it to a file. All *Effects* in the model are collected into a single ontology that contains all *OWL Named Individuals*, *OWL Classes*, *OWL Object Properties*, *OWL Data Properties* and *OWL Assertions*. This ontology will naturally be inconsistent but it is a valuable resource for the knowledge engineers maintaining the background knowledge base. The ontology contains information about which expert to contact, the business analyst that designed the process and the process model from which the ontology was generated. A knowledge engineer can view the process model from the online repository to get a contextual overview of how terms are used. They can directly contact the expert or business analyst with questions about the semantics of the terms used and the terms are already in the correct format for inclusion into the background knowledge base. These rudimentary ontologies can be directly opened in Protege [108].

6.3.5 Accumulator Class

The *Accumulator* class contains all the methods used in accumulation. See chapter 3.6.2 for further details about the functions or appendix A for the actual algorithms. The class also contains a method for walking through a process model accumulating *Effect Scenarios* as it goes. This is the standard method of accumulation that offers real-time processing. See chapter 3.6.1 for more information about the difference between accumulation procedures. The more complex and thorough accumulation procedure requests a *Scenario Label* from the *Scenario Label* class and will accumulate up to the selected activity in the process model. The *Scenario Label* method will generate all possible *Cumulative Effect Scenarios* for the selected activity no matter where it occurs in the process.

6.3.6 Scenario Label Class

The *Scenario Label* class assembles a *Scenario Label* (see chapter 3.6.1) for each *ProcessNode* in the process model. The class contains all the methods for walking through a process model and mapping model elements into partially ordered sets (posets). A *Scenario Label* is a poset that resembles an execution path through the process model from the start event to a specific *ProcessNode*. However, it is not

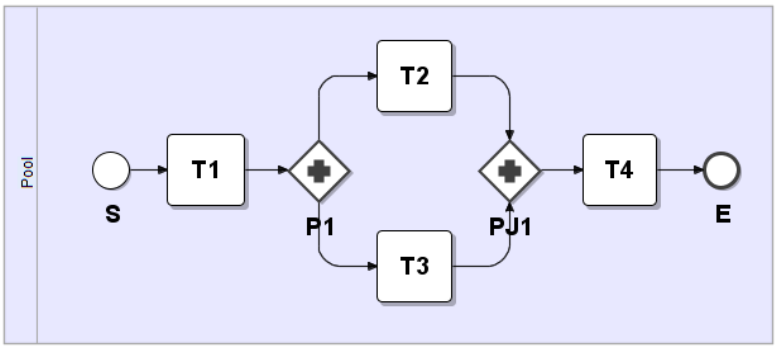

Figure 6.9: A basic BPMN model showing a parallel gateway structure.

simply a path. The nature of the poset offers instructions to the *Accumulator* class about whether to use a set of accumulation algorithms or a set of joining algorithms to derive a *Cumulative Effect Scenario*.

Scenario Label processing can take considerably longer than standard accumulation so it is offered to the user as an option. The standard mode of accumulation will correctly compute all possible worlds in existence at any *ProcessNode* outside a parallel or inclusive *Gateway Structure* but it will only compute the possible worlds for *ProcessNodes* inside the *Gateway Structure* as if from the perspective of an individual agent executing a task on a single parallel branch. The agent is unaware of what happens on other parallel branches. This is not a serious impediment thanks to conflict catching techniques described in chapter 3.6.1.

A *Scenario Label* for the entire process model in Figure 6.9 is

$\langle S, T1, \{T2, T3\}, T4, E\rangle$.

However, when calculating the *Cumulative Effect Scenarios* at $T2$, two *Scenario Labels* would be used:

$\langle S, T1, T2\rangle$

$\langle S, T1, \{T2, T3\}\rangle$

Either of these two paths could have executed but in the second instance $T2$ and $T3$ are joined/merged before accumulating with $T1$. The braces { } indicate a merging operation rather than an accumulation operation.

6.4 ProcessSEER Accumulation Implementation

This section describes the components involved in accumulation and the interactions between them. Two key components in the implementation that are not mentioned in chapter 3 are the *Process Node Queue* and the *Process Node List*. They control the order in which *ProcessNodes* are accumulated.

6.4.1 Process Node Queue

Each BPMN element in a process model is contained in a node, an instance of a *ProcessNode* class. A queue is used to schedule the accumulation of nodes. Pairwise accumulation always processes two nodes, the current node and the previous node. The current node contributes an *Immediate World List* and the previous node contributes a *Cumulative World List* (see chapter 3.5). Nodes are added to the queue as they are discovered during model traversal. Each node is popped off the beginning of the queue and its previous node is tested to determine whether it has a *Cumulative World List*. If the previous node contains a *Cumulative World List* then it has already been accumulated and the current node, that was popped off the queue, can now be accumulated. Accumulated nodes are added to a *Process Node List* to indicate they have already been accumulated. If the previous node does not contain a *Cumulative World List* then it has not been accumulated yet so the current node is returned to the end of the queue. Accumulation of the entire process model is complete when the queue is empty.

6.4.2 Process Node List

The *Process Node List* is used primarily to prevent unhandled looping. The ProcessSEER tool has a limited capability for processing loops. The tool therefore detects when a loop occurs by checking whether it has already processed a *ProcessNode*. When a *ProcessNode* has been accumulated it is moved from the *Process Node Queue* to the *Process Node List*. If the *Accumulator* walk-through method encounters a *ProcessNode* that is in the *Process Node List* then it processes the accumulation as if the previous *ProcessNode* was an exclusive gateway split (see chapter A.9). *Condition Scenarios* are annotated to the looping sequence flow entering the activity. If any one of the alternate *Cumulative Effect Scenarios* entering from the loop does not satisfy the *Condition Scenario* then the loop stops. In some cases looping will produce identical *Effect Scenarios* every time the loop executes because of the nature of the state descriptions. In such cases accumulation can be stopped manually if the business analyst has neglected to include some type of incremental *Effect*. Other times may require quantitative data to be updated or, as is usual, a looping sequence of activities changes the world description in such a way that it affects the outcome of the first activity in the loop. BPMN has a number of different looping structures that include events and subprocesses that are not supported by the ProcessSEER tool. In its current version the tool only supports sequence flow looping like the example shown in Fig:6.10. The sequence flow between $T3$ and $T1$ will contain a *Condition Scenario* annotation.

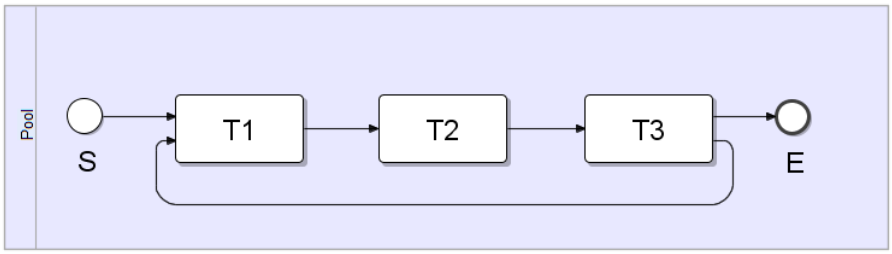

Figure 6.10: A BPMN looping structure supported by the ProcessSEER tool.

6.4.3 Components Involved in Accumulation

There are 18 components involved in semantic effect accumulation:

- Process Nodes (see Ch 6.3.1)
- Effects (see Ch 6.1)
- Effect Scenarios (see Ch 6.3.2)
- Ancestor Sequences (see Ch 6.6)
- Conditions (see Ch 6.1)
- Condition Scenarios (see Ch 6.3.3)
- Immediate World Lists (see Ch 6.5)
- Cumulative World Lists (see Ch 6.5)
- A Process Node Queue (see Ch 6.4.1)
- A Processed Node List (see Ch 6.4.2)
- Gateway Splits (see Ch 6.8)
- Gateway Joins (see Ch 6.8)
- Branch Clusters (see Ch 3.8.10)
- Branch Groups (see Ch 3.8.12)
- Branch Combinations (see Ch 6.7)
- Effect Scenario Ontologies (see Ch 6.3.2)
- Background Knowledge Base (see Ch 6.3.4)
- A Reasoning Engine (see Ch 6.3.4)

The current version of ProcessSEER has only limited functionality and will only accumulate process models that contain a single start event. The ProcessSEER tool currently only supports accumulation of the following BPMN elements:

- Tasks
- Empty Start, Intermediate and End Events
- Inclusive, Exclusive and Parallel Gateways

An instance of a *ProcessNode* class (see 6.3.1) is created for each of the above BPMN elements. An instance of a *ProcessNode* is referred to simply as a node. ProcessSEER must first locate the start event. It does this by walking backward through the process model until it reaches the start event. The start event is then added to the *Process Node Queue* (see 6.4.1) also referred to simply as a queue. A queue is used to hold nodes that have not been accumulated.

The first step in accumulation is to load all the nodes into the queue. During accumulation a *Cumulative World List* is created for each node (see chapter 3.5). Each node is selected from the beginning of the queue and its previous nodes (previous within the model not the queue) are checked to see if they contain a *Cumulative World List*. If all previous nodes contain a *Cumulative World List* then the selected node is accumulated otherwise it is returned to the end of the queue to be processed later. Once accumulated the node is removed from the queue and added to the *Process Node List*. This process continues until all nodes have been accumulated.

There are three cases to consider when checking whether a node is ready to accumulate, no preceding nodes, one preceding node or many preceding nodes. To be considered ready to be accumulated, all preceding nodes must contain a *Cumulative World List*. If the node is ready to be accumulated then it is passed to an *Accumulator* class (see 6.3.5). This class contains all the different accumulation functions and determines which function is required. The *Accumulator* algorithms can be found in appendix A. Accumulation is dependent on three things:

1. The number of preceding nodes.
2. The type of the current node.
3. The type of preceding node/s.

If the current node has no preceding nodes then it is considered to be a start event. Node types are checked and if they are supported then they are accumulated otherwise the user is issued with a warning dialog. If the current node has only one preceding node then the current node can be anything but a gateway join. If the current node has more than one preceding node then it is considered to be a gateway

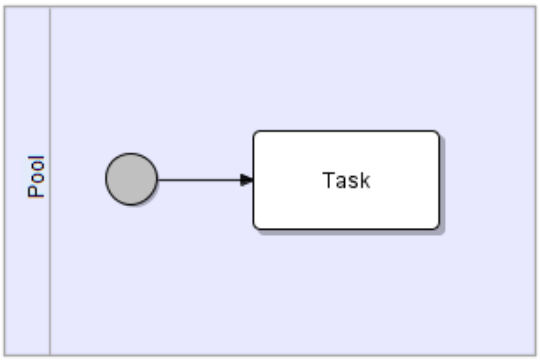

Figure 6.11: A BPMN fragment with the start event highlighted. A start event has no preceding nodes.

join. The tool will also process informal joins. The following subsections refer to processing the current node depending on the number of preceding nodes. Highlighted BPMN elements within the figures indicate the current node being accumulated.

6.4.4 Processing Zero Preceding Nodes

In the case of a start event (Fig: 6.11) there is no previous *Cumulative World List* to accumulate. The *Immediate World List* of a start event may also be empty. In this case an empty *Cumulative World List* is added to the start event node so that the next activity can be accumulated. The start event is a doorway into the process model through which external world descriptions can be added. The *Cumulative World List* of another process model can be added as the *Cumulative World List* of a start event to see how the process will operate under those conditions. It is, however, more common to add individual *Cumulative Effect Scenarios* to the start event *Cumulative World List* as these represent a single outcome of a previous process model.

Sometimes business analysts will establish starting *Effects* by annotating them to the start event. In this case the *Immediate World List* of the start event is duplicated and becomes the *Cumulative World List*. Once the *Cumulative World List* has been computed it is stored in the node. The node is then removed from the queue and stored in the *Process Node List*. The next node in the queue is then assessed. This may not be the next node in the process which is why its previous nodes are checked for whether they contain a *Cumulative World List*.

6.4.5 Processing a Single Preceding Node

A current node with only one preceding node is the most common form of accumulation. Within this scope there are still a number of different accumulation methods

node	S	A	X	P	I	XJ	PJ	IJ	L
S→	n/a	pci	pc	pc	pc	n/a	n/a	n/a	pci
A→	n/a	pi	p	p	p	p	b	b	pci
X→	n/a	pci	pc	pc	pc	pc	n/a	n/a	pci
P→	n/a	pi	p	p	p	n/a	n/a	n/a	pci
I→	n/a	pci	pc	pc	pc	n/a	n/a	pc	pci
XJ→	n/a	pi	p	p	p	p	b	b	pci
PJ→	n/a	pi	p	p	p	p	b	b	pci
IJ→	n/a	pi	p	p	p	p	b	b	pci
L→	n/a	pci	p	p	p	n/a	n/a	n/a	pci

Table 6.1: Required elements for accumulation.

used to accommodate the different BPMN elements. Table 6.1 shows the required elements for accumulation to occur. The four main elements needed are the previous *Cumulative World List*, the *Immediate World List*, a *Condition Scenario* and a *Branch Combination*. *Cumulative* and *Immediate World Lists* consist of *Ancestor Sequences* which contain different versions of the same *Effect Scenario*. *Condition Scenarios* consist of *Conditions* (syntactically identical the an *Effect*) and *Branch Combinations* consist of *Ancestor Sequences* that share the same *Effect Scenario History*. The nodes heading each column in Table 6.1 indicate the current node that is being accumulated whereas the nodes in each row represent the node immediately preceding the current node. For example, an activity preceded by an exclusive gateway split will require a previous *Cumulative World List* from the gateway, a *Condition Scenario* from the sequence flow between the gateway and the activity, and the *Immediate World List* from the activity.

There are 17 different algorithms involved in accumulating the limited number of BPMN elements supported by the ProcessSEER tool (see appendix A). Different pairings of BPMN elements require different algorithms. The *acc*( ) function referred to in [63] is far more complex in practice than in theory. The following Table 6.1 identifies the different pairings of *ProcessNodes* and the elements required to accumulate them. Each pairing of a current and previous node will require either a previous *Cumulative Effect Scenario*, an *Immediate Effect Scenario*, a *Condition Scenario* or a *Branch Combination* or combinations of these elements.

From Table 6.1 it can be seen that there are five different element combinations `p, pi, pc, pci, b` each warranting a unique style of accumulation.

p-type Accumulations

Pairings that require only the previous *Cumulative Effect Scenario* occur most commonly when the *ProcessNode* being accumulated is a gateway split. Note that any

Table 6.2: Table 6.1 Legend

S	Start Event
A	Activity
X	Exclusive Gateway Split
P	Parallel Gateway Split
I	Inclusive Gateway Split
XJ	Exclusive Gateway Join
PJ	Parallel Gateway Join
IJ	Inclusive Gateway Join
L	Looping Activity
→	A previous node
p	previous Cumulative World List
c	Condition Scenario
i	Immediate World List
b	Branch Combinations
n/a	Not Applicable

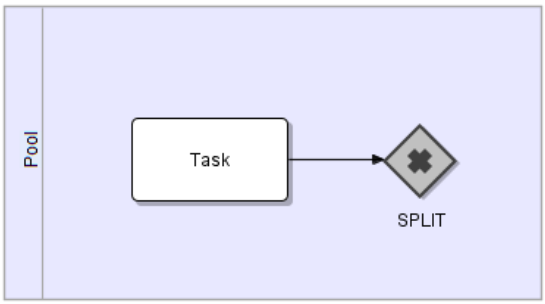

Figure 6.12: A pairing showing an exclusive gateway split being accumulated with a previous task.

reference to a *Cumulative Effect Scenario* implies that it exists inside an *Ancestor Sequence* which in turn exists inside a *Cumulative World List*. If all that is needed is a previous *Cumulative Effect Scenario* then the *ProcessNode* being accumulated does not contain an *Immediate World List*. In the case of an exclusive gateway split (see Fig:6.12) the previous *Cumulative World List* is simply copied as the current *Cumulative World List* of the XOR split. However, in the case of a parallel or inclusive gateway split (see Figures 6.13 and 6.14) the top *Cumulative Effect Scenario* of each *Ancestor Sequence* in the previous *Cumulative World List* is duplicated and added back to the top of each *Stack*. The *Cumulative World List* is copied but all *Ancestor Sequences* within contain a copy of the top *Cumulative Effect Scenario* (see Fig:6.15).

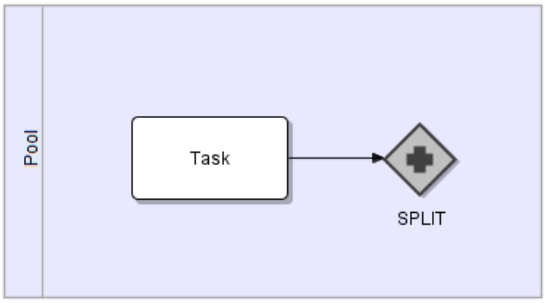

Figure 6.13: A pairing showing a parallel gateway split being accumulated with a previous task.

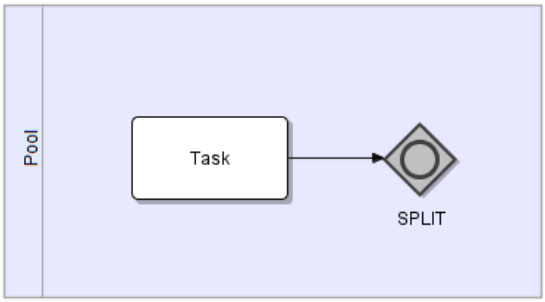

Figure 6.14: A pairing showing an inclusive gateway split being accumulated with a previous task.

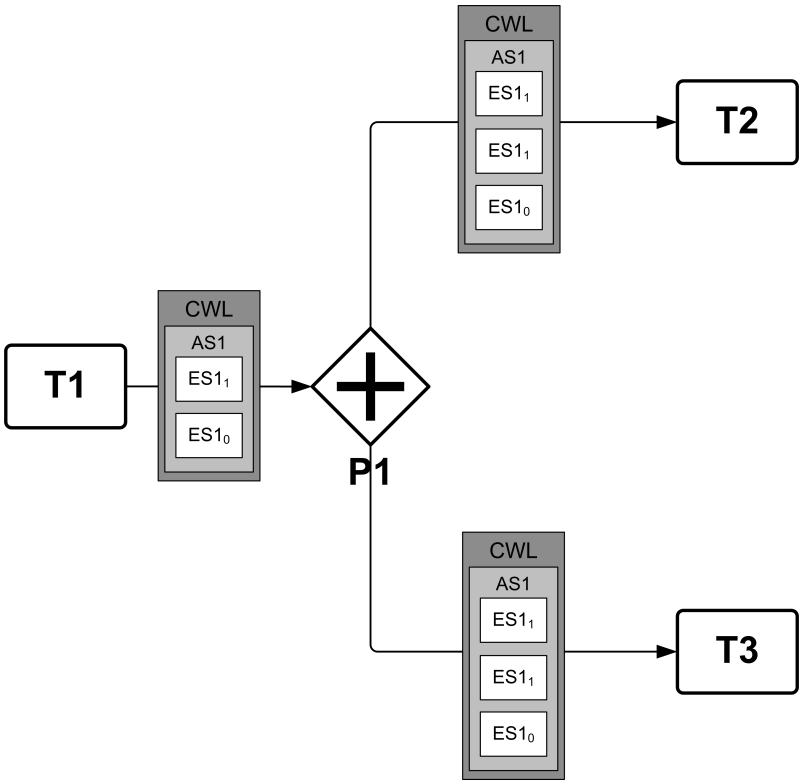

Figure 6.15: A parallel gateway split showing how the top Cumulative Effect Scenario is duplicated in an Ancestor Sequence and how the Cumulative World List is duplicated for each branch.

pi-type Accumulations

Pairings that require both a previous *Cumulative World List* and an *Immediate World List* are the most common. This particular pairing requirement satisfies the criteria for accumulation as presented in [63, 64]. A previous *Cumulative Effect Scenario* from the top of an *Ancestor Sequence Stack* inside a previous *Cumulative World List* is accumulated with an *Immediate Effect Scenario* from inside an *Ancestor Sequence Stack* inside an *Immediate World List* (the verbose description). All this manipulation of *Ancestor Sequences* and *World Lists* only confuses the explanation so let us focus only on the accumulation of *Effect Scenarios*. The pair-wise accumulation is typically $cCES = acc(pCES, IES)$ where $pCES$ is the previous *Cumulative Effect Scenario*, $cCES$ is the current or resulting *Cumulative Effect Scenario* and IES is the *Immediate Effect Scenario*.

The *Immediate Effect Scenario* is a statement of fact, i.e., every *Effect* is known to be a direct post-condition of an action being executed. All *Immediate Effects* therefore take priority over any previous *Cumulative Effects*. Inconsistencies are resolved by passing the union of the two *Effect Scenarios* to the *Possible Worlds* function (see appendix A.3). The potential exists for this function to increase the number of *Effect Scenarios* generated but in practice it was found that it rarely returns more than one *Effect Scenario* as a result. The maximal subset criteria applied by the function minimises the number of returned results.

The accumulation implementation utilises the fields from the individual *Effects* and the *Effect Scenario Ontology* (ESO). When the previous *Cumulative Effect Scenario* and the *Immediate Effect Scenario* are unioned the *Effects* are merged into a single *Effect Scenario*. A new ESO is then generated from the *Effects*. The *Possible Worlds* function does not add or remove single previous *Cumulative Effects*. It tests complete subsets of the original inconsistent union. The outcome of sequentially adding or removing previous *Cumulative Effects* to the *Immediate Effect Scenario* can be influenced by the sequential order in which the *Effects* are added or removed. The *Effect Scenarios* returned from the *Possible Worlds* function are the result of this type of accumulation.

pc-type Accumulations

Pairings that require only the previous *Cumulative World List* and a *Condition Scenario* occur when the *ProcessNode* being accumulated has no *Immediate World List*. This is most common when the node being accumulated is a gateway and the previous node is an exclusive or inclusive gateway split. However, it is possible that an activity will have no annotations. *Ancestor Sequences* in the previous *Cumulative World List* simply become *Ancestor Sequences* in the resulting *Cumulative World*

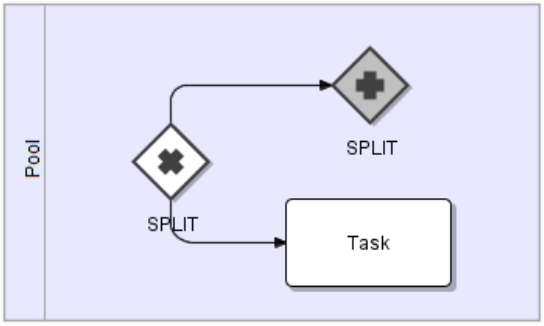

Figure 6.16: A pairing showing a parallel gateway split being accumulated with a previous exclusive gateway split.

List iff they satisfy the *Conditions* in the *Condition Scenario*. This type of accumulation only utilises the *Decision Function* (see chapter 3.6.3 or A.1 to view the algorithm). Figure 6.16 shows a nested parallel gateway split immediately following and exclusive gateway split. The parallel gateway split has no *Immediate World List* but the previous *Cumulative World List* from the exclusive gateway split must be filtered with the *Condition Scenario* annotated to the sequence flow.

pci-type Accumulations

Pairings that require the previous *Cumulative World List*, a *Condition Scenario* and an *Immediate World List* most commonly occur when the previous *ProcessNode* is an exclusive or inclusive gateway split (see Fig:6.17). They also occur when the previous node is the last node in a looping structure or the previous node is a start event. Only *Ancestor Sequences* whose top *Cumulative Effect Scenario* satisfies the *Condition Scenario* qualify for inclusion into a subset of the previous *Cumulative World List*. This subset becomes the previous *Cumulative World List* for the node being accumulated. In this case the original previous *Cumulative World List* is passed to the *Decision Function* whose output is then passed along with the *Immediate World List* to the *Pair-wise Accumulation Function* which in turn utilises the *Possible Worlds Function* (see appendix A.9).

6.4.6 Processing Multiple Preceding Nodes

Multiple preceding nodes always occur at gateway joins. Exclusive gateway joins only require the previous *Cumulative World Lists* from each preceding node (see Fig:6.18). The *Ancestor Sequences* from each branch *Cumulative World List* are simply combined into a single resulting *Cumulative World List*. Each *Ancestor Sequence* is representative of a unique possible world. However, parallel and inclusive gateway joins require *Branch Combinations* to be accumulated.

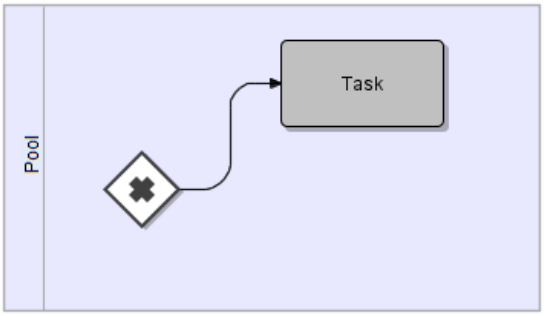

Figure 6.17: A pairing showing a task being accumulated with a previous exclusive gateway split.

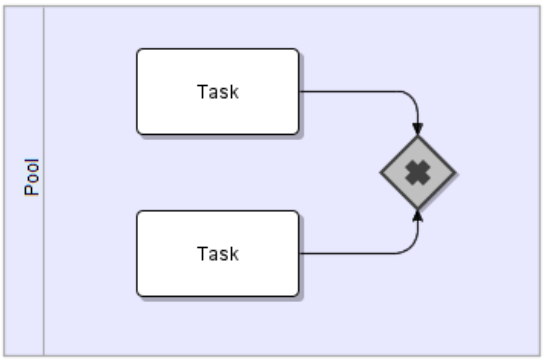

Figure 6.18: A pairing showing an exclusive gateway join being accumulated with two previous tasks.

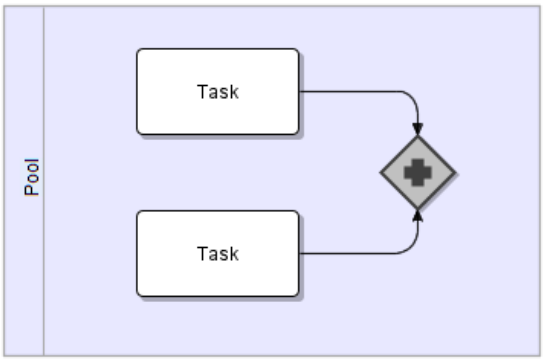

Figure 6.19: A pairing showing a parallel gateway join being accumulated with two previous tasks.

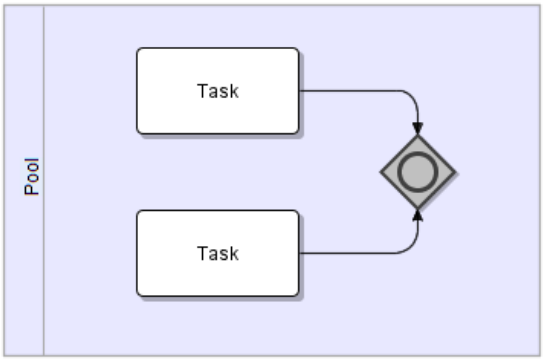

Figure 6.20: A pairing showing an inclusive gateway join being accumulated with two previous tasks.

b-type Accumulations

Pairings that require *Branch Combinations* occur when the *ProcessNode* being accumulated is a parallel or inclusive gateway join. Figures 6.19 and 6.20 show the typical accumulation pairs. The *Ancestor Sequences* in each previous *Cumulative World List* must be assembled into multiple *Branch Combinations*. This is a complex procedure involving multiple functions (see chapter 3.8.9). *Ancestor Sequences* and *Branch Combinations* play an integral part in join accumulation. The implementation of these components is described in the following sections.

6.5 World List Implementation

World Lists do not require a specialised class. They are implemented as standard Java List objects because they are only containers for *Ancestor Sequences* implemented as *Stacks*. Each *ProcessNode* contains at most one *Immediate World List* and one *Cumulative World List*. The *Ancestor Sequences* in an *Immediate World List* contain the annotated *Effect Scenarios* entered by the business analyst while the *Ancestor Sequences* in a *Cumulative World List* contain accumulated *Effect Scenarios*. A *World List* is representative of all the possible worlds that can be derived from a single action. Figure 6.21 shows the structure of a *World List*. Every world (top Effect Scenario in each Ancestor Sequence) is accumulated separately.

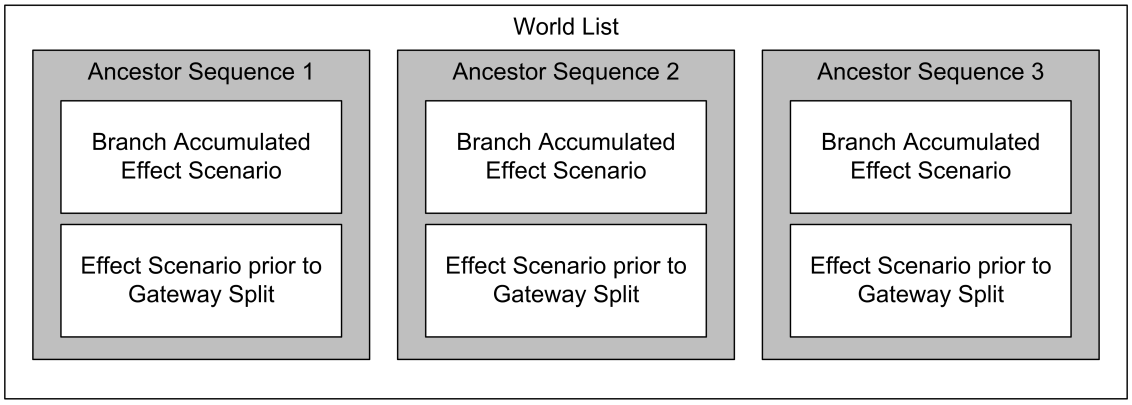

Figure 6.21: Structure of a World List containing branch Stacks of Effect Scenarios (Ancestor Sequences).

6.6 Ancestor Sequence Implementation

An *Ancestor Sequence* is implemented as a *Stack*, i.e., the last *Effect Scenario* added to the *Stack* is always the first *Effect Scenario popped off* the *Stack*. A *Stack* is used to store versions of an *Effect Scenario* (see chapter 3.7.2). The top version on the *Stack* is acted upon during accumulation. All versions underneath the top *Effect Scenario* are referred to as the *Effect Scenario History*. It is used to track an *Effect Scenario* when it is copied and propagated along different branches of a *Gateway Structure*. Consider a *Stack* containing a single *Effect Scenario*. When it enters a parallel *Gateway Structure* the *Effect Scenario* in the *Stack* is duplicated and placed on top of the *Stack*. The *Stack* now contains two identical *Effect Scenarios* (see Fig:6.15). The *Stack* is copied and propagated along each branch. The top version will be accumulated along each branch while the versions underneath remain unchanged. The version immediately underneath represents the *Cumulative Effect Scenario* as it was when it entered the *Gateway Structure* and can be used to identify each *Ancestor Sequence's* origin after the top version has been altered by different parallel activities. *Ancestor Sequences* from different branches that have the same *Effect Scenario Histories* are collected into *Branch Combinations* before merging can occur. *Branch Combinations* contain only one matching *Ancestor Sequence* from each branch.

6.7 Branch Combination Implementation

A *Branch Combination* is a grouping of *Ancestor Sequences* from different branches in a *Gateway Structure*. *Branch Combinations* are generated at the end of the *Gateway Structure* as part of the gateway join accumulation. *Branch Combinations* contain *Stacks* of *Cumulative Effect Scenarios*, i.e.,*Ancestor Sequences*. Each *Stack*

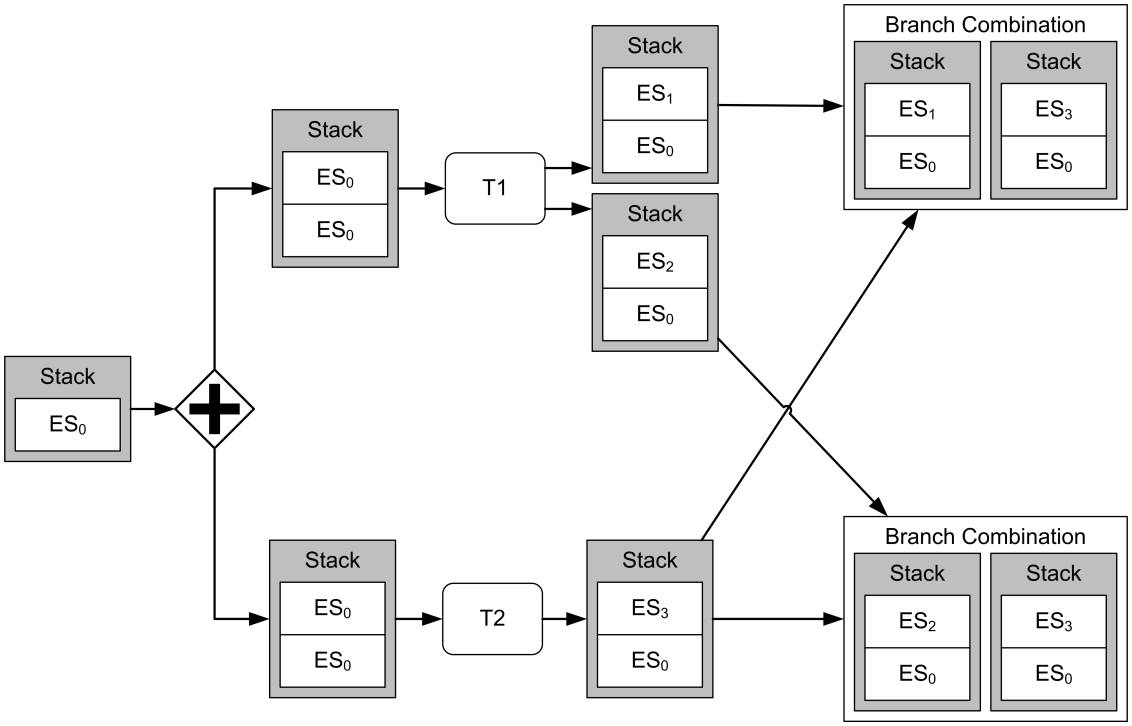

Figure 6.22: Diagram showing how Branch Combinations are generated before a parallel gateway join.

in a *Branch Combination* has originated from the same pre-*Gateway Structure Stack*, i.e., each *Ancestor Sequence* has the same *Effect Scenario History*. In Fig.6.22 it can be seen how *Branch Combinations* are generated at the end of a parallel *Gateway Structure*. The *Stack* containing the *Effect Scenario* (ES_0) enters the parallel gateway split and is copied for each branch. ES_0 is also duplicated and added to the top of each Stack. The top *Effect Scenarios* are altered by accumulation with the *Immediate Effect Scenarios* of $T1$ and $T2$ to produce ES_1, ES_2 and ES_3. ES_0 is still present underneath each of these new *Effect Scenarios* indicating that they all originated from the same *Effect Scenario*. At the parallel gateway join, one *Stack* from each branch is matched according to its originating *Effect Scenario* and grouped into a *Branch Combination*.

Branch Combinations are only generated at a gateway join and are treated differently for each type of gateway. Exclusive gateway joins do not generate *Branch Combinations* because *Effect Scenarios* are never split across branches. They therefore do not need to be reassembled at a gateway join. A *Branch Combination* of a parallel gateway join will contain exactly the same number of *Stacks* as there are incoming branches. A *Branch Combination* of an inclusive gateway join will contain a maximum of one *Stack* for each incoming branch. Whether an *Effect Scenario* is propagated along any particular branch of an inclusive *Gateway Structure* is determined by the *Condition Scenarios* for each branch.

6.8 Processing Gateway Structures

A *Gateway Structure* encompasses all activities and gateways that occur between a gateway split and a gateway join. *Gateway Structures* are treated as if they were a subprocess (see chapter 3.7). *Effect Scenarios* can become split into multiple copies of themselves when passing through a *Gateway Structure*. It therefore becomes necessary to track the progress of each *Effect Scenario* as it travels through the *Gateway Structure*. Tracking is achieved through the use of *Ancestor Sequences* (see chapter 3.7.2). For convenience the following paragraphs summarise what is described in more detail in chapter 3.7.1. Let us consider each *Gateway Structure* separately.

A *Cumulative World List* entering a parallel gateway split will be directed along all branches so it is copied. Each copy contains *Ancestor Sequences* whose *Effect Scenario Histories* link them across branches. *Effect Scenario Histories* capture the state of an *Effect Scenario* before it was split across different parallel branches. Parallel *Gateway Structures* are the reason possible worlds need to be tracked because when the possible worlds reach a parallel gateway join, they need to be matched with their copies from other branches. Possible worlds from each branch that have the same origin are grouped into what is referred to as a *Branch Combination*. In this way a single possible world can be acted upon by different activities at the same time then reassembled to reveal the complete outcome. Possible worlds are matched at the parallel gateway join and collected into *Branch Combinations*. A *Branch Combination* for a parallel gateway join contains exactly one matching possible world from each branch. Reassembling takes the top *Effect Scenario* (possible world) off each *Stack* (Ancestor Sequence) and merges them together into a single Effect Scenario. This union of top *Effect Scenarios* is used as the *Immediate Effect Scenario* for the *Gateway Structure*. The *Effect Scenario Histories* of all *Ancestor Sequences* in the *Branch Combination* will be identical. The *Effect Scenario* directly underneath the top represents the previous *Cumulative Effect Scenario* prior to the *Gateway Structure*. The union of top *Effect Scenarios* is therefore accumulated with this previous *Cumulative Effect Scenario* to determine the outcome of the entire *Gateway Structure* on the previous *Cumulative Effect Scenario*.

In contrast a possible world entering an exclusive gateway split will be directed along only one branch so it will only ever be sequentially accumulated. When it reaches the exclusive gateway join, it will have no correspondence to any possible world on any other branch. Therefore, possible worlds from every branch of an exclusive *Gateway Structure* are all disjunctive. An exclusive gateway join therefore collects all possible worlds from every branch into a single *Cumulative World List*. There is no need for *Branch Combinations* because each *Branch Combination* will

only contain a single possible world because each possible world was never split so no merging is required.

A possible world entering an inclusive gateway split will be directed along zero or more branches depending on whether it satisfies one or more sets of *Conditions*. Within an inclusive gateway structure, activities can either be executed in isolation or in parallel depending on the current state of the world. Whenever a possible world satisfies multiple *Condition Scenarios* it is copied to each satisfied conditional branch. In this way it is divided and must be reassembled at the gateway join. The difference here is that a possible world can travel along any number of branches depending on whether it satisfies the *Conditions* of the branch. When it comes to reassembling a possible world, not all branches will contain an accumulated version of an original possible world. Once again, tracking is necessary to identify which possible worlds from each branch should be merged. *Branch Combinations* from an inclusive *Gateway Structure* will contain only one possible world from each branch but not all branches may be represented. The *Conditions* will determine which possible world is copied over multiple branches and which is propagated over only one branch. Some may not satisfy any conditions although this would be unusual and should prompt the business analyst to investigate the anomaly.

6.9 Future Implementations

This section discusses intended features for the ProcessSEER tool. BPMN applies an XML structure to all process models that can be easily transformed from one XML Schema to another. The use case diagram in Fig:6.23 shows a number of collaborative use cases that involve the exchange of information between different professionals in an organisation. Greyed out use cases indicate that the feature is not yet fully functional. Currently, the exchange of information involves requirements engineering to translate between the different vocabularies used by these professionals. The ProcessSEER framework is being designed to alleviate many of the problems associated with translating between business and technical jargon. The tool is being designed to generate skeleton code for semantic web services based on the artifacts that are input into and output from an activity. The states described in *Effect* annotations can be used as pre- and post-conditions in SAWSDL files providing a richer description of the web service that automated online agents can reason with. Automated generation of skeleton code can include input parameters and expected outputs. The automated ontology generation has already significantly improved knowledge acquisition efforts for knowledge engineers and it is hoped that semantic web service support will be equally well received.

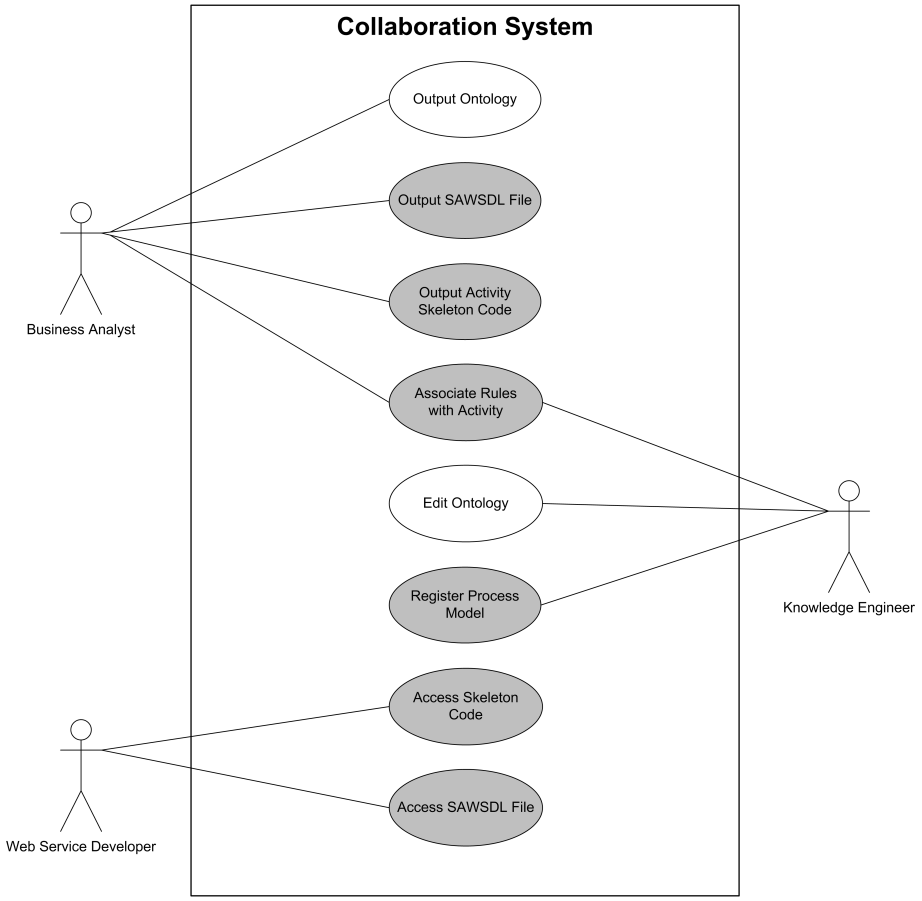

Figure 6.23: A use case diagram showing existing and intended features to support collaboration.

Unfortunately at the time of writing this document the SAWSDL standard had not fully implemented preconditions and effects [85] even though they had featured in the first semantic web service proposals like OWL-S [96], WSMO [27] and WSDL-S [5]. However, SAWSDL does offer some basic support for describing conditions. The following snippet from the SAWSDL User Guide [129] shows how conditions can be applied to both input and output. One *inputRule* and three *outputRules* are applied as *modelReferences*.

```
<wsdl:description...>
  <wsdl:interface name="CheckAvailabilityRequestService">
   <wsdl:operation name="CheckAvailabilityRequestOperation"
            pattern="http://www.w3.org/ns/wsdl/in-out">
     <wsdl:input element="
       CheckAvailabilityRequestServiceRequest"
       sawsdl:modelReference="http://org1.example.com/rules#
          inputRule"/>
     <wsdl:output element="
        CheckAvailabilityRequestServiceResponse"
       sawsdl:modelReference="http://org1.example.com/rules#
```

```
          outputRule1
               http://org1.example.com/rules#outputRule2
               http://org1.example.com/rules#outputRule3"/>
   </wsdl:operation>
 </wsdl:interface>
</wsdl:description>
\caption{\textbf{Snippet from a SAWSDL file showing operational
   conditions associated with a web service's input and output
   .}}\label{SAWSDL}
```

The *outputRules* referred to in the above SAWSDL snippet are:

outputRule1 If more than 50lbs was ordered, order is taken only for 50lbs

outputRule2 ItemConfirmation # can be tracked only for 30 days after the order was placed

outputRule3 If the order does not arrive at the shipping location within the required date, no charge will be made to the account

Note that the conditions referred to in [129] do not refer to the *Conditions* defined in this document. Conditions in SAWSDL refer, in this case, to rules which in our framework are contained within an external rules base or within an external artifact repository. The rules to which the modelReferences refer include *Conditions* but a rule is structurally different from a *Condition* within the ProcessSEER framework. The following example shows how outputRule1 is expressed within the ProcessSEER framework:

$$hasWeightInPoundsGreaterThan(requestedOrder, 50) \rightarrow$$
$$hasWeightInPounds(acceptedOrder, 50)$$

Rules are expressed as inferences that are constructed from *Conditions/Effects* which are constructed from concepts and predicates in the background ontology. This ensures a common vocabulary is being used across an organisation. However, interoperability between organisations will require mapping between ontologies.

6.10 Summary

This chapter describes how *Effects* have been transformed from sentences in Controlled Natural Language (CNL) to SPOTONS that comply with international standards like OWL and RDF. The ProcessSEER tool is described including the development platform and the open source modelling tool on which it is based. Examples

are given of the graphic user interface that describe its functionality. The implementation of the conceptual components described in chapter 3 have also been detailed along with some components unique to the implementation. This includes how the Web Ontology Language (OWL) has been utilised for knowledge representation. The types of accumulations of BPMN elements has also been explained and the future direction of the application has been outlined. Within its limited scope the ProcessSEER application has demonstrated the efficacy of the framework underpinning it. A complete listing of the ProcessSEER source code can be found at `http://www.dsl.uow.edu.au/projects/processSEER`

Chapter 7

Conclusion

7.1 Research Questions

7.1.1 RQ1

Is it possible to provide richer semantics for business process models via effect annotations?

Business process models provide little by way of semantic description of the effects of a process beyond what can be conveyed via the nomenclature of tasks and the decision conditions associated with gateways. A framework for annotating semantic effects to business process models has been described in chapter 3 and shown by example that it is possible to provide richer semantics to a process model via effect annotations. Annotated models have been shown to reveal previously unrecognised consequences providing new insights and improved model design.

7.1.2 RQ2

Is it possible to determine, at design time, for any point in a process design, the effects that a process would achieve if it were to execute up to that point?

The framework supports automated reasoning allowing it to determine what would happen if a process model were to execute up to a particular point. Two methods of effect accumulation offer a choice between real-time accumulation or, computing a complete answer to this question. The use of Scenario Labels offers the complete answer to the question, "What would have happened if the process had executed up to this point?", but the computational cost can be excessive in some circumstances.

7.1.3 RQ3

Is it possible to build a robust tool to support business analysts in developing such annotated process models?
The ProcessSEER tool described in chapter 6 provides support to business analysts for manually annotating effects to process models. The tool includes a user friendly interface requiring no formal language skills to operate it. The design of the tool has been influenced by reviews from business analysts so as to appeal to the skill set of its users.

7.1.4 RQ4

Is it possible to support analysis tasks such as goal satisfaction, compliance checking, semantic conformance checking and semantic simulation?
The ProcessSEER tool allows annotated effects to be accumulated through a model providing a semantic simulation environment. Goal satisfaction and compliance checking can be achieved by specifying goals or compliance criteria in the vocabulary prescribed by the domain ontology and comparing them with the outcomes of process activities. Although the tool allows freedom of data entry, it contributes to a background ontology which ultimately controls semantic conformance. The tool can be instrumental in developing a standardised domain ontology.

7.1.5 RQ5

Is it possible to crowd source these annotations?
Effect annotations supplied by a single business analyst only offer a limited perspective on reality whereas multiple perspectives provide a much richer body of knowledge. Crowdsourcing techniques that can be utilised to provide multi-perspective views of process activities are described in chapter 5. Techniques for assimilating and analysing crowdsourced information are also provided.

7.1.6 RQ6

Do compelling use cases for this technology exist in use cases such as medicine?
Chapter 4 explores use cases in the clinical domain and found significant benefit from the use of semantically annotated process models. The most compelling case involves the detection of conflicts between co-incident treatment protocols (in the same patient). Comparing semantically annotated process models of treatment protocols can reveal these conflicts thus avoiding potentially damaging consequences.

7.2 Contributions to the Research Community

The academic contribution of this research is a framework for the specification of semantic effects that supports automated reasoning of the outcomes of activities in business process models. The conceptual framework has been proven to be effective by way of an implemented tool that facilitates the annotation of semantic effects to business process models and performs real-time accumulation reasoning of complex activity structures.

7.3 Contributions to the Practitioner Community

It is my hope that the ProcessSEER tool can be utilised by practitioners to provide richer semantics to process models allowing those models to be utilised in new and exciting ways. The tool's ability to output rudimentary ontologies makes it an important contributor in the area of knowledge acquisition. Ultimately it is my hope that the tool will provide enough incentive to motivate the business community to become more active in the development of the semantic web.

Bibliography

[1] Wil M. P. van der Aalst. "The Application of Petri Nets to Workflow Management". In: *Journal of Circuits, Systems and Computers* 08.01 (1998), pp. 21–66. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218126698000043.

[2] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. "Business Process Management: A Survey". In: *Business Process Management: International Conference, BPM 2003 Eindhoven, The Netherlands, June 26–27, 2003 Proceedings*. Ed. by Wil M. P. van der Aalst and Mathias Weske. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–12. ISBN: 978-3-540-44895-2. URL: http://dx.doi.org/10.1007/3-540-44895-0_1.

[3] W.M.P. van der Aalst. "Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype". In: *BPTrends* 3.5 (May 2004), pp. 1–11. URL: http://is.tm.tue.nl/research/patterns/download/bptrendsPiHype.pdf.

[4] C.J. Acuna and E. Marcos. "Modeling semantic web services: a case study". In: *Proceedings of the 6th international conference on Web engineering (ICWE 06)*. Vol. 263. New York, NY USA: ACM, July 2006, pp. 32–39. ISBN: 1-59593-352-2.

[5] Rama Akkiraju et al. *Web Service Semantics - WSDL-S*. Tech. rep. IBM, Jan. 2006. URL: http://domino.research.ibm.com/library/cyberdig.nsf/papers/EF9FE52551FB21DC8525710D005A8480/$File/rc23854.pdf.

[6] R. Akkiraju et al. *Web Service Semantics - WSDL-S*. World Wide Web Consortium. 2005. URL: http://www.w3.org/Submission/WSDL-S/.

[7] R. Altwarg. "Controlled Languages - An Introduction". MA thesis. Sydney: Macquarie University, 2000.

[8] Grigoris Antoniou and Frank van Harmelen. "Web Ontology Language: OWL". In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 67–92. ISBN: 978-3-540-24750-0. URL: http://dx.doi.org/10.1007/978-3-540-24750-0_4.

[9] Rainer Anzbock and Schahram Dustdar. "Modeling Medical E-services". In: *Business Process Management*. Vol. 3080/2004. Heidelberg, Berlin: Springer, June 2004, pp. 49–65. ISBN: 978-3-540-22235-4. URL: http://www.infosys.tuwien.ac.at/Staff/sd/papers/ModelingMedicale-services_final.pdf.

[10] Franz Baader and Bernhard Hollunder. "A terminological knowledge representation system with complete inference algorithms". In: *Processing Declarative Knowledge: International Workshop PDK '91 Kaiserslautern, Germany, July 1–3, 1991 Proceedings*. Ed. by Harold Boley and Michael M. Richter. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 67–86. ISBN: 978-3-540-46667-3. URL: http://dx.doi.org/10.1007/BFb0013522.

[11] Franz Baader, Ian Horrocks, and Ulrike Sattler. "Chapter 3 Description Logics". In: *Handbook of Knowledge Representation*. Ed. by Vladimir Lifschitz Frank van Harmelen and Bruce Porter. Vol. 3. Foundations of Artificial Intelligence. Elsevier, May 2008, pp. 135–179. URL: http://www.sciencedirect.com/science/article/pii/S1574652607030039.

[12] Franz Baader and Werner Nutt. "Basic Description Logics". In: *The Description Logic Handbook*. Ed. by Franz Baader et al. New York, NY, USA: Cambridge University Press, 2003, pp. 43–95. ISBN: 0-521-78176-0. URL: http://dl.acm.org/citation.cfm?id=885746.885749.

[13] Jie Bao et al. *OWL 2 Web Ontology Language Quick Reference Guide (Second Edition)*. Tech. rep. W3C, Dec. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/.

[14] Charlton Barreto et al. *Web Services Business Process Execution Language Version 2.0 Primer*. Tech. rep. Oasis, May 2007. URL: http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf.

[15] Dave Beckett. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[16] D.A. Bensaber and M. Malki. "Development of semantic web services: Model Driven Approach". In: *Proceedings of the 8th international conference on New technologies in distributed systems (NOTERE 08)*. 40. Lyon, France: ACM, 2008. ISBN: 978-1-59593-937-1.

[17] T. Berners-Lee, J. Hendler, and O. Lassila. "The Semantic Web". In: *Scientific American Magazine* (May 2001). URL: http://www.sciam.com/article.cfm?id=00048144-10D2-1C70-84A9809EC588EF21&page=1.

[18] Tim Berners-Lee. *HTML Tags*. Web Page. Nov. 1992. URL: https://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html.

[19] Tim Berners-Lee. *Re: status. Re: X11 BROWSER for WWW*. Internet Correspondence. Oct. 1991. URL: http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html.

[20] Harold Boley. "Relationships between Logic Programming and RDF". In: *Advances in Artificial Intelligence. PRICAI 2000 Workshop Reader: Four-Workshops held at PRICAI 2000 Melbourne,Australia,August 28 - September 1, 2000 Revised Papers*. Ed. by Ryszard Kowalczyk et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 201–218. ISBN: 978-3-540-45408-3. URL: http://dx.doi.org/10.1007/3-540-45408-X_21.

[21] Alex Borgida. "On the relative expressiveness of description logics and predicate logics". In: *Artificial Intelligence* 82.01 (1996), pp. 353–367. ISSN: 0004-3702. URL: http://www.sciencedirect.com/science/article/pii/0004370296000045.

[22] Aziz A Boxwala et al. "GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines". In: *Journal of Biomedical Informatics* 37.3 (2004), pp. 147–161. ISSN: 1532-0464. URL: http://www.sciencedirect.com/science/article/pii/S1532046404000334.

[23] R. Brachman. "A Structural Paradigm for Representing Knowledge". PhD thesis. USA: Harvard University, 1977.

[24] Ronald J. Brachman and James G. Schmolze. "An overview of the KL-ONE Knowledge Representation System". In: *Cognitive Science* 09.02 (1985), pp. 171–216. ISSN: 0364-0213. URL: http://www.sciencedirect.com/science/article/pii/S0364021385800148.

[25] M. Brambilla et al. "Model-Driven Design and Development of Semantic Web Service Applications". In: *ACM Transactions on Internet Technology* 8.1 (Nov. 2007).

[26] Jeen Broekstra et al. "Enabling knowledge representation on the web by extending RDF schema". In: *Computer networks* 39.5 (2002), pp. 609–634.

[27] Jos de Bruijn et al. *Web Service Modeling Ontology (WSMO)*. W3C Submission. W3C, June 2005. URL: https://www.w3.org/Submission/WSMO/.

[28] Avik Chaudhuri et al. "EON: Modeling and analyzing dynamic access control systems with logic programs". In: *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 381–390.

[29] Peter P. Chen. "Entity-Relationship Diagrams and English Sentence Structure". In: *Proceedings of the 1st International Conference on the Entity-Relationship Approach to Systems Analysis and Design*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1980, pp. 13–14. ISBN: 0-444-85487-8. URL: http://dl.acm.org/citation.cfm?id=647508.726196.

[30] Peter Pin-Shan Chen. "The Entity-relationship Model&Mdash;Toward a Unified View of Data". In: *ACM Trans. Database Syst.* 1.1 (Mar. 1976), pp. 9–36. ISSN: 0362-5915. URL: http://doi.acm.org/10.1145/320434.320440.

[31] David Cohn and Richard Hull. "Business artifacts: A data-centric approach to modeling business operations and processes". In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 32.3 (2009), pp. 3–9. URL: http://researcher.watson.ibm.com/researcher/files/us-hull/2009-09-cohn-hull-on-artifact-centric-research-IEEE-Data-Eng-Bull-preprint.pdf.

[32] E. Coiera. "Clinical Decision Support Systems". In: *The Guide to Health Informatics*. 2nd. London, UK: Arnold, Oct. 2003. Chap. 25. ISBN: 0340 764252. URL: http://www.coiera.com/aimd.htm.

[33] R. Curia, L. Gallucci, and M. Ruffolo. "Knowledge Management in Health Care: an Architectural Framework for Clinical Process Management Systems". In: *Proceedings of the Sixteenth International Workshop on Database and Expert Systems Applications*. Copenhagen, Denmark: IEEE, Aug. 2005, pp. 393–397. ISBN: 0-7695-2424-9.

[34] J. Curry, C. McGregor, and S. Tracey. "A Communication Tool to Improve the Patient Journey Modeling Process". In: *Proceedings of the 28th IEEE EMBS Annual International Conference*. New York City, USA, Aug. 2006.

[35] Randall Davis, Howard Shrobe, and Peter Szolovits. "What Is a Knowledge Representation?" In: *AI Magazine* 14.1 (1993), pp. 17–33. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1029/947.

[36] A Dwivedi et al. "Workflow management systems: the healthcare technology of the future?" In: *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*. IEEE, 2001, pp. 3887–3890. ISBN: 0-7803-7211-5.

[37] *Eclipse SOA Tools Platform, BPMN Modeller*. 2009. URL: http://www.eclipse.org/bpmn/.

[38] J. Edelman et al. "Tangible Business Process Modelling: A New Approach". In: *Proceedings of ICED 09, the 17th International Conference on Engineering Design, Vol. 6, Design Methods and Tools (pt. 2)*. Ed. by M. Norell Bergendahl et al. Vol. 6. Stanford University, Palo Alto, California: Design Society, Aug. 2009, p. 485. ISBN: 978-1-904670-10-0. URL: https://www.designsociety.org/publication/28763/tangible_business_process_modeling.

[39] D. Fensel et al. "OIL: an ontology infrastructure for the Semantic Web". In: *IEEE Intelligent Systems* 16.2 (Mar. 2001), pp. 38–45. ISSN: 1541-1672.

[40] Richard E. Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*. London, England: Morgan Kaufmann Publishers Inc., Aug. 1971, pp. 608–620.

[41] H. Foster et al. "Leveraging Eclipse for integrated model-based engineering of web service compositions". In: *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. New York, NY, USA: ACM, 2005, pp. 95–99.

[42] John Fox, Nicky Johns, and Ali Rahmanzadeh. "Disseminating medical knowledge: the PROforma approach". In: *Artificial intelligence in medicine* 14.1 (1998), pp. 157–182.

[43] N.E. Fuchs, U. Schwertel, and R. Schwitter. "Attempto Controlled English: Not Just Another Logic Specification Language". In: *LOPSTR'98, LNCS 1559*. Berlin Heidelberg: Springer-Verlag, 1999, pp. 1–20.

[44] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. "Attempto Controlled English for Knowledge Representation". In: *Reasoning Web: 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*. Ed. by Cristina Baroglio et al. Berlin, Heidelberg: Springer

Berlin Heidelberg, 2008, pp. 104–124. ISBN: 978-3-540-85658-0. URL: http://dx.doi.org/10.1007/978-3-540-85658-0_3.

[45] Norbert E. Fuchs and Rolf Schwitter. "Web-Annotations for Humans and Machines". In: *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*. Lecture Notes in Computer Science. Springer, 2007.

[46] Norbert E. Fuchs et al. "Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines". In: *Reasoning Web: First International Summer School 2005, Msida, Malta, July 25-29, 2005, Revised Lectures*. Ed. by Norbert Eisinger and Jan Maluszynski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 213–250. ISBN: 978-3-540-31675-6. URL: http://dx.doi.org/10.1007/11526988_6.

[47] Henry L. Gantt. "A Graphical Daily Balance in Manufacture". In: *Transactions of the American Society of Mechanical Engineers*. Vol. 24. New York City: The American Society of Mechanical Engineers, 1903, pp. 1322–1336. URL: https://hdl.handle.net/2027/mdp.39015023119541?urlappend=%3Bseq=1358.

[48] Peter Gardenfors. *Belief Revision*. Vol. 29. Cambridge University Press, 2003.

[49] Michael Gelfond and Vladimir Lifschitz. "Action Languages". In: *Electronic Transactions on AI* 2 (1998), pp. 193–210. URL: http://www.ep.liu.se/ej/etai/1998/007/.

[50] Guiseppe Getto. *ASD-STE100 Simplified Technical English*. Tech. rep. Belgium, Jan. 2013. URL: http://guiseppegetto.com/pwr393/wp-content/uploads/2013/02/ASD-STE100-ISSUE-6.pdf.

[51] A.K. Ghose and G. Koliadis. "Business Process Compliance: Techniques for Design-Time Auditing and Resolution". In: *Proceedings of the First International Workshop on Juris-informatics (JURISIN07)*. 2007.

[52] A.K. Ghose and G. Koliadis. "PCTk: A ToolKit for Managing Business Process Compliance". In: *Proceedings of the Second International Workshop on Juris-informatics (JURISIN'08)*. 2008.

[53] Frank Bunker Gilbreth and Lillian Moller Gilbreth. "Process charts-first steps in finding the one best way". In: *American Society of Mechanical Engineers (ASME)* (1921).

[54] Matthew L. Ginsberg and David E. Smith. "Reasoning about Action I - A Possible Worlds Approach". In: *Artificial Intelligence* 35.2 (1988), pp. 165–195. ISSN: 0004-3702. URL: http://www.sciencedirect.com/science/article/pii/0004370288900112.

[55] Guido Governatori et al. "Detecting Regulatory Compliance of Business Process Models through Semantic Annotations". In: *Proceedings of the 4th International Workshop on Business Process Design*. 2008.

[56] F.B. Green. "Managing the unmanagable: Integrating the supply chain with new developments in software". In: *Supply Chain Management: An International Journal* 6.5 (2001), pp. 208–211.

[57] P. Green, M. Rosemann, and M. Weske. "Integrated Process Modeling: An Ontological Evaluation". In: vol. 25. 2. Jan. 2000, pp. 73–87.

[58] Nicola Guarino, Daniel Oberle, and Steffen Staab. "What Is an Ontology?" In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17. ISBN: 978-3-540-92673-3. URL: http://dx.doi.org/10.1007/978-3-540-92673-3_0.

[59] Ramanathan Guha and Dan Brickley. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[60] C. Gutierrez, C. A. Hurtado, and A. Vaisman. "Introducing Time into RDF". In: *IEEE Transactions on Knowledge and Data Engineering* 19.2 (Feb. 2007), pp. 207–218. ISSN: 1041-4347. URL: http://www.cs.rpi.edu/~willig4/temp/time.pdf.

[61] C. Hall and P. Harmon. *The 2005 Enterprise Architecture, Process Modeling & Simulation Tools Report*. Tech. rep. 2005. URL: http://www.bptends.com.

[62] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems - Theory and Practice*. Springer US, 2010, pp. 9–22. ISBN: 978-1-4419-5652-1. URL: http://www.springer.com/cda/content/document/cda_downloaddocument/9781441956521-c1.pdf.

[63] K. Hinge, A. Ghose, and G. Koliadis. "Process SEER: A Tool for Semantic Effect Annotation of Business Process Models". In: *Proceedings of the Thirteenth IEEE International EDOC Conference*. Auckland, NZ: IEEE, Sept. 2009.

[64] Kerry Hinge, Aditya Ghose, and Andrew Miller. "A Framework for Detecting Interactions Between Co-Incident Clinical Processes". In: *International Journal of E-Health and Medical Communications* 1.2 (2010), pp. 24–35.

[65] K. Hoesch-Klohe and A. K. Ghose. "Business Process Improvement in Abnoba". In: *SEE 2010. Proceedings of the 1st International Workshop on Services, Energy and Ecosystem*. Dec. 2010. URL: http://greenprocess.org/articles/.

[66] Joerg Hoffmann, Ingo Weber, and Frank Kraft. *SAP Speaks PDDL: Exploiting a Software-Engineering Model for Planning in Business Process Management*. 2014. URL: https://www.jair.org/media/3636/live-3636-6417-jair.pdf.

[67] D. Hollingsworth. *The Workflow Reference Model*. Tech. rep. Hampshire, UK, Jan. 1995. URL: http://www.wfmc.org/standards/docs/tc003v11.pdf.

[68] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for OWL Ontologies". In: *Semantic Web* 2.1 (2011), pp. 11–21. ISSN: 1570-0844. URL: http://dl.acm.org/citation.cfm?id=2019470.2019471.

[69] I. Horrocks. "Semantic Web: The Story So Far". In: *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*. Vol. 225. Banff, Canada: ACM, 2007, pp. 120–125. ISBN: 1-59593-590-8. URL: http://doi.acm.org/10.1145/1243441.1243469.

[70] Ian Horrocks and Peter Patel-Schneider. "Reducing OWL entailment to description logic satisfiability". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.4 (2004). International Semantic Web Conference 2003, pp. 345–357. ISSN: 1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826804000095.

[71] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. "From SHIQ and RDF to OWL: the making of a Web Ontology Language". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1.1 (2003), pp. 7–26. ISSN: 1570-8268. URL: http://www.sciencedirect.com/science/article/pii/S1570826803000027.

[72] I. Horrocks et al. *The Ontology Inference Layer OIL*. Tech. rep. Vrije Universiteit Amsterdam, NL., Aug. 2000. URL: http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2000/oil.pdf.

[73] Richard Hull. "Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges". In: *On the Move to Meaningful Internet Systems: OTM 2008: OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II*. Ed. by Robert Meersman and Zahir Tari. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1152–1163.

ISBN: 978-3-540-88873-4. URL: http://dx.doi.org/10.1007/978-3-540-88873-4_17.

[74] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[75] Kaoru Ishikawa. *What is total quality control? The Japanese way*. Prentice Hall, 1985.

[76] Daniel Jackson and Janette Wing. "Lightweight Formal Methods". In: *IEEE Computer* (1996), pp. 21–22.

[77] Kaarel Kaljurand. "Attempto Controlled English as a Semantic Web Language". PhD thesis. Faculty of Mathematics and Computer Science, University of Tartu, 2007.

[78] H. Katsuno and A. O. Mendelzon. "On the Difference Between Updating a Knowledge Base and Revising It". In: *Belief Revision*. Cambridge University Press, 1992, pp. 183–203.

[79] Graham Klyne and Jeremy Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[80] G. Koliadis and A. Ghose. "Correlating Business Process and Organizational Models to Manage Change". In: *Australasian Conference on Information Systems*. Adelaide, SA, Australia, Dec. 2006.

[81] G. Koliadis and A. Ghose. "Verifying Semantic Business Process Models in Inter-operation". In: *SCC 2007. IEEE International Conference on Services Computing, 2007.* Auckland, NZ: IEEE, July 2007, pp. 731–738.

[82] G. Koliadis, A. K. Ghose, and S. Padmanabhuni. "Towards an Enterprise Business Process Architecture Standard". In: *Proceedings of the IEEE Services Congress Symposium on SOA Standards 2008*. Hawaii, US: IEEE, July 2008.

[83] G. Koliadis et al. "A Combined Approach for Supporting the Business Process Model Lifecycle". In: *Proc. of the 10th Pacific Asia Conference on Information Systems (PACIS'06)*. 2006.

[84] J. Kopecky. *Aligning WSMO and WSDL-S*. 2005. URL: http://www.wsmo.org/TR/d30/v0.1/20050805/d30v01_20050805.pdf.

[85] Jacek Kopecky. *Re: Preconditions and Effects in SAWSDL?* 2007. URL: https://lists.w3.org/Archives/Public/public-sws-ig/2007Mar/0012.html.

[86] J. Krogstie, C. Veres, and G. Sindre. "Integrating Semantic Web Technology, Web Services, and Workflow Modeling: Achieving System and Business Interoperability". In: *International Journal of Enterprise Information Systems* 8.1 (Jan. 2007), pp. 22–41.

[87] Markus Krotzsch, Frantisek Simancik, and Ian Horrocks. "A Description Logic Primer". In: *CoRR* abs/1201.4089 (2012). URL: http://arxiv.org/abs/1201.4089.

[88] Tri A. Kurniawan et al. "On Formalizing Inter-process Relationships". In: *Business Process Management Workshops*. Ed. by Florian Daniel et al. Vol. 100. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, Dec. 2012, pp. 75–86. ISBN: 978-3-642-28115-0. URL: http://dx.doi.org/10.1007/978-3-642-28115-0_8.

[89] Ora Lassila. *Resource Description Framework (RDF) Model and Syntax Specification*. (W3C) Recommendation. W3C, Feb. 1999. URL: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[90] Lance Laytner. *Star Trek Tech*. Web Page. 2009. URL: https://web.archive.org/web/20111028200606/http://www.editinternational.com/read.php?id=4810edf3a83f8.

[91] Paolo Liberatore. "Belief Merging by Examples". In: *ACM Trans. Comput. Logic* 17.2 (Dec. 2015), 9:1–9:38. ISSN: 1529-3785. URL: http://doi.acm.org/10.1145/2818645.

[92] Y. Lin. "Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability". PhD thesis. Trondheim, Norway: Norwegian University of Science and Technology, Mar. 2008.

[93] A. T. Manes. *Web Services: A Manager's Guide*. Boston, MA: Addison Wesley, June 2003. ISBN: 0-321-18577-3.

[94] Frank Manola and Eric Miller. *RDF Primer*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[95] David Martin and John Domingue. "Semantic Web Services, Part 2". In: *IEEE Intelligent Systems* November/December (2007), pp. 8–15.

[96] D. Martin et al. *OWL-S: Semantic Markup for Web Services*. Tech. rep. Nov. 2004. URL: http://www.w3.org/Submission/OWL-S/.

[97] D. Martin et al. "Semantic Web Services, Part 2". In: *IEEE Intelligent Systems* 22.6 (Nov. 2007), p. 8.

[98] Philippe Andre Martin. "Knowledge representation in RDF/XML, KIF, Frame-CG and Formalized-English". In: *Proceedings of the 1st International Semantic Web Conference ISWC 2002*. Sardinia, Italy, July 2002. URL: http://www.phmartin.info/webKB/doc/papers/iccs02/iswc02.pdf.

[99] K. Maximini and M. Schaaf. "The PROGEMM Approach For Managing Clinical Processes". In: *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Copenhagen, Denmark: IEEE, June 2003, pp. 332–337. ISBN: 0-7695-1963-6.

[100] W. McCune. *Prover9*. 2009. URL: http://www.cs.unm.edu/~mccune/prover9/.

[101] D. Mcdermott et al. *PDDL - The Planning Domain Definition Language*. Tech. rep. TR-98-003. Yale Center for Computational Vision and Control, 1998. URL: http://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf.

[102] Deborah McGuinness. "Description Logics Emerge from Ivory Towers". In: *The First Semantic Web Working Symposium (SWWS2001)*. 2001. URL: http://data.semanticweb.org/conference/iswc/2001/position-proceedings/paper-31.

[103] Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language Overview*. World Wide Web Consortium, Recommendation REC-owl-features-20040210. Feb. 2004. URL: http://www.w3.org/TR/2004/REC-owl-features-20040210.

[104] Shelia A. McIlraith, Tran Cao Son, and Honglei Zen. "Semantic Web Services". In: *IEEE Intelligent Systems* March/April (2001), pp. 46–53.

[105] Marvin Minsky. *A Framework for Representing Knowledge*. Tech. rep. Cambridge, MA, USA, 1974. URL: http://web.media.mit.edu/~minsky/papers/Frames/frames.html.

[106] E. D. Morrison et al. "Strategic Alignment of Business Processes". In: *WESOA'11. Proceedings of the 7th International Workshop on Engineering Service Oriented Applications*. Paphos, Cyprus, Dec. 2011. URL: http://www.fnord.be/research/publications.

[107] Nataliya Mulyar et al. "Declarative and procedural approaches for modelling clinical guidelines: Addressing flexibility issues". In: *Business Process Management Workshops: BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane, Australia, September 24, 2007, Revised Selected Papers*. Ed. by Arthur ter Hofstede, Boualem Benatallah, and Hye-Young Paik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008,

pp. 335–346. ISBN: 978-3-540-78238-4. URL: http://dx.doi.org/10.1007/978-3-540-78238-4_35.

[108] Natalya F Noy et al. "Protege-2000: an open-source ontology-development and knowledge-acquisition environment". In: *AMIA Annu Symp Proc*. Vol. 953. 2003, p. 953. URL: http://ejournal.narotama.ac.id/files/an%5C%20open-source%5C%20ontology-development%5C%20and%5C%20knowledge-acquisition%5C%20environment.pdf.

[109] D. Oberle et al. "Supporting Application Development in the Semantic Web". In: *ACM Transactions on Internet Technology* 5.2 (May 2005), pp. 328–358.

[110] OMG. *BPMN 2.0 by Example*. Object Management Group. June 2010. URL: http://http://www.omg.org/spec/BPMN/20100601/10-06-02.pdf.

[111] OMG. *Business Process Model and Notation (BPMN) Specification 2.0.2*. Object Management Group. Feb. 2013. URL: http://www.omg.org/spec/BPMN.

[112] OMG. *Business Process Modeling Notation (BPMN) 1.1 Specification*. Object Management Group. Feb. 2006. URL: http://www.omg.org/spec/BPMN.

[113] OMG. *Object Constraint Language (OCL) Specification 2.4*. Object Management Group. Feb. 2014. URL: http://www.omg.org/spec/OCL/.

[114] OMG. *OMG Unified Modeling Language (OMG UML), Infrastructure*. Object Management Group. Feb. 2009. URL: http://www.omg.org/spec/UML/2.2/.

[115] Silvia Panzarasa et al. "Evidence-based careflow management systems: the case of post-stroke rehabilitation". In: *Journal of Biomedical Informatics* 35.2 (Apr. 2002), pp. 123–139.

[116] Jean Paoli, Tim Bray, and Michael Sperberg-McQueen. *(XML) 1.0 Recommendation*. (W3C) Recommendation. W3C, Feb. 1998. URL: http://www.w3.org/TR/1998/REC-xml-19980210.

[117] Bijan Parsia et al. *OWL 2 Web Ontology Language Primer (Second Edition)*. Tech. rep. W3C, Dec. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[118] Peter Patel-Schneider and Boris Motik. *OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)*. W3C Recommendation. W3C, Dec. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/.

[119] Edwin P. D. Pednault. "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus". In: *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. Toronto, Canada: Morgan Kaufmann Publishers Inc., Aug. 1989, pp. 324–332. ISBN: 1-55860-032-9.

[120] M.K. Raut and A. Singh. "Prime Implicates of First Order Formulas". In: *International Journal of Computer Science and Applications* 1 (2004).

[121] Jan C. Recker et al. "Do Process Modelling Techniques Get Better? A Comparative Ontological Analysis of BPMN". In: *16th Australasian Conference on Information Systems*. Ed. by Bruce Campbell, Jim Underwood, and Deborah Bunker. Sydney, Australia: Australasian Chapter of the Association for Information Systems, 2005. URL: http://eprints.qut.edu.au/2879/.

[122] Jan C. Recker et al. "How Good is BPMN Really? Insights from Theory and Practice". In: *14th European Conference on Information Systems*. Ed. by Jan Ljungberg and Magnus Andersson. Goeteborg, Sweden: Australasian Chapter of the Association for Information Systems, 2006. URL: http://eprints.qut.edu.au/4636/.

[123] Hajo A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Berlin, Heidelberg: Springer-Verlag, 2003. ISBN: 3-540-01186-2.

[124] Wolfgang Reisig. *Petri Nets: An Introduction*. New York, NY, USA: Springer-Verlag New York, Inc., 1985. ISBN: 0-387-13723-8.

[125] Raymond Reiter. "The Frame Problem in Situation the Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression". In: *Artificial Intelligence and Mathematical Theory of Computation*. Ed. by Vladimir Lifschitz. San Diego, CA, USA: Academic Press Professional, Inc., 1991, pp. 359–380. ISBN: 0-12-450010-2. URL: http://dl.acm.org/citation.cfm?id=132218.132239.

[126] D. Roman et al. "Web Service Modeling Ontology". In: *Applied Ontology* 1.1 (2005), pp. 77–106.

[127] Michael Rosemann. "The Service Portfolio of a BPM Center of Excellence". In: *Handbook on Business Process Management 2: Strategic Alignment, Governance, People and Culture*. Ed. by Jan vom Brocke and Michael Rosemann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 267–284. ISBN: 978-3-642-01982-1. URL: http://dx.doi.org/10.1007/978-3-642-01982-1_13.

[128] M. Ruffolo et al. "Semantic Clinical Process Management". In: *Proceedings of the Twentieth IEEE International Symposium on Computer-Based Medical Systems*. Washington DC, USA: IEEE, 2007, pp. 518–523. ISBN: 0-7695-2905-4.

[129] Brahmananda Sapkota and Rama Akkiraju. *Semantic Annotations for WSDL and XML Schema - Usage Guide*. W3C Note. W3C, Aug. 2007. URL: http://www.w3.org/TR/2007/NOTE-sawsdl-guide-20070828/.

[130] J. Schulz-Hofen and S. Golega. "Generating web applications from process models". In: *Workshop proceedings of the sixth international conference on Web engineering (ICWE 06)*. Vol. 155. 6. Palo Alto, CA, USA: ACM, 2006, p. 6. ISBN: 1-59593-435-9.

[131] R. Schwitter. *Home (Controlled Natural Languages)*. 2009. URL: http://sites.google.com/site/controllednaturallanguage/.

[132] R. Schwitter and N.E. Fuchs. "Attempto - From Specifications in Controlled Natural Language towards Executable Specifications". In: *CoRR* cmp-lg/9603004 (May 1996).

[133] Rolf Schwitter. "Controlled Natural Languages for Knowledge Representation". In: *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. COLING '10. Beijing, China: Association for Computational Linguistics, 2010, pp. 1113–1121. URL: http://dl.acm.org/citation.cfm?id=1944566.1944694.

[134] Rolf Schwitter. "Working for Two: A Bidirectional Grammar for a Controlled Natural Language". In: *AI 2008: Advances in Artificial Intelligence: 21st Australasian Joint Conference on Artificial Intelligence Auckland, New Zealand, December 1-5, 2008. Proceedings*. Ed. by Wayne Wobcke and Mengjie Zhang. Berlin Heidelberg: Springer-Verlag, 2008, pp. 168–179. ISBN: 978-3-540-89378-3. URL: http://dx.doi.org/10.1007/978-3-540-89378-3_17.

[135] J. Scott and G. Marshall. *A Dictionary of Sociology*. New York, NY: Oxford University Press, Apr. 2005. URL: http://www.oxfordreference.com/views/ENTRY.html?subview=Main%5C&entry=t88.e1610.

[136] Andreas Seyfang, Robert Kosara, and Silvia Miksch. *Asbru's Reference Manual, Asbru Version 7.3*. Tech. rep. Asgaard-TR-2002-1. Vienna University of Technology, Institute of SoftwareTechnology, Vienna, Jan. 2002. URL: https://www.researchgate.net/publication/277291122_Asbru's_Reference_Manual_Asbru_Version_73.

[137] Murray Shanahan. *Solving the Frame Problem - A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[138] H. Sharp et al. "The role of story cards and the wall in XP teams: a distributed cognition perspective". In: *AGILE 2006 (AGILE'06)*. IEEE, July 2006, pp. 11–75.

[139] Rob Shearer, Boris Motik, and Ian Horrocks. "HermiT: A Highly-Efficient OWL Reasoner". In: *OWLED 2008, Fifth International Workshop*. Vol. 432. Karlsruhe, Germany, Oct. 2008, p. 91. URL: http://webont.org/owled/2008/papers/owled2008eu_submission_12.pdf.

[140] Herbert Alexander Simon. *The Sciences of the Artificial*. MIT Press, 1996. ISBN: 978-0-2621-9374-0. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/0201379406.

[141] Pnina Soffer. "Scope analysis: identifying the impact of changes in business process models". In: *Software Process: Improvement and Practice* 10.4 (2005), pp. 393–402.

[142] Pnina Soffer and Yair Wand. "Goal-Driven Analysis of Process Model Validity". In: *Lecture Notes in Computer Science* 3084 (June 2004), pp. 521–535.

[143] John F Sowa. "Semantic Networks". In: *Encyclopedia of Cognitive Science*. John Wiley & Sons, Ltd, 2006. ISBN: 9780470018866. URL: http://dx.doi.org/10.1002/0470018860.s00065.

[144] M. Stefanelli. *Careflow Management Systems*. Commissioned Briefing Paper, Online. 2002. URL: http://www.openclinical.org/briefingpaperStefanelhtml.

[145] Frederick W. Taylor. *The Principles of Scientific Management*. New York and London: Harper & Brothers, 1911. URL: https://archive.org/stream/principlesofscie00taylrich#page/n0/mode/2up.

[146] J.T.E. Timm and G.C. Gannod. "A Model-Driven Approach for Specifying Semantic Web Services". In: *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*. IEEE, July 2005, pp. 313–320.

[147] Dmitry Tsarkov and Ian Horrocks. "DL Reasoner vs. First-Order Prover". In: *Proceedings of the 2003 Description Logic Workshop*. Vol. 81. 2003, pp. 152–159. URL: http://ceur-ws.org/Vol-81/tsarkov.pdf.

[148] C. Lee Ventola. "Mobile Devices and Apps for Health Care Professionals: Uses and Benefits". In: *Pharmacy and Therapeutics* 39.5 (2014), pp. 356–364. ISSN: 1052-1372.

[149] Jos B. Warmer and Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling With Uml (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, Oct. 1998. ISBN: 0201379406. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/0201379406.

[150] Ingo Weber, Jorg Hoffman, and Jan Mendling. "Semantic Business Process Validation". In: *Proceedings of the 3rd International Workshop on Semantic Business Process Management*. 2008.

[151] I. Weber et al. "Towards a Methodology for Semantic Business Process Modeling and Configuration". 2004. URL: http://www.sti-innsbruck.at/fileadmin/documents/papers/joerg_hoffmann/semsoc07.pdf.

[152] S. Weerawarana et al. *Web Services Platform Architecture*. New Jersey, NJ: Pearson Education, Inc., Sept. 2005.

[153] S. A. White and D. Miers. *BPMN Modeling and Reference Guide*. Lighthouse Pt, FL: Future Strategies Inc., Sept. 2008. URL: http://www.futstrat.com/books/BPMN-Guide.php.

[154] S.A. White. *Using BPMN to Model a BPEL Process*. BPTrends. Mar. 2005. URL: http://homepages.borland.com/jkaster/callisto/reference/%20MappingBPELtoBPMN.pdf.

[155] James M. Wilson. "Gantt charts: A centenary appreciation". In: *European Journal of Operational Research* 149.2 (2003), pp. 430–437. ISSN: 0377-2217. URL: http://www.sciencedirect.com/science/article/pii/S0377221702007695.

[156] Sira Yongchareon and Chengfei Liu. "A process view framework for artifact-centric business processes". In: *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*. Springer, 2010, pp. 26–43. URL: http://www.ict.swin.edu.au/personal/syongchareon/papers/A%5C%20Process%5C%20View%5C%20Framework%5C%20%5C%20for%5C%20Artifact-Centric%5C%20Business%5C%20Processes.pdf.

Appendix A

Algorithms

Algorithm A.1 Decision Function

Let KBR be a consistent background knowledge base and set of rules.
Let $pCES$ be a previous *Cumulative Effect Scenario*.
Let pWL be a previous *World List* of previous *Cumulative Effect Scenarios*.
Let $cCES$ be a current *Cumulative Effect Scenario*.
Let cWL be a current *World List* of current *Cumulative Effect Scenarios*.
Let CS be a *Condition Scenario*.
function DEC(pWL, CS)
    $cWL = \emptyset$
    $cCES = \emptyset$
    for all ($pCES \in pWL$) **do**
        if ($CS \subseteq pCES$) **then**
            $cCES = pCES$
            cWL . ADD($cCES$)
        end if
    end for
    return cWL
end function

Algorithm A.2 Combinatorial Function

Let IES be an *Immediate Effect Scenario.*
Let iWL be an immediate *World List* of *Immediate Effect Scenarios.*
Let $pCES$ be a previous *Cumulative Effect Scenario.*
Let pWL be a previous *World List* of previous *Cumulative Effect Scenarios.*
Let $cCES$ be a current *Cumulative Effect Scenario.*
Let cWL be a current *World List* of current *Cumulative Effect Scenarios.*
Let $Pair$ be a sequence of exactly two *Effect Scenarios* $\langle pCES, IES \rangle$
Let PS be a set of $Pairs$.

```
function COMBO(pWL, iWL)
    PS = ∅
    for all (pCES ∈ pWL) do
        Pair = ∅
        for all (IES ∈ iWL) do
            Pair = ⟨pCES, IES⟩
            PS . ADD(Pair)
        end for
    end for
    return PS
end function
```

Algorithm A.3 Possible Worlds Function

Let KBR be a consistent background knowledge base and set of rules.
Let IES be an *Immediate Effect Scenario*.
Let iWL be an immediate *World List* of *Immediate Effect Scenarios*.
Let $pCES$ be a previous *Cumulative Effect Scenario*.
Let pWL be a previous *World List* of previous *Cumulative Effect Scenarios*.
Let $cCES$ be a current *Cumulative Effect Scenario*.
Let cWL be a current *World List* of current *Cumulative Effect Scenarios*.
Let $Pair$ be a sequence of exactly two *Effect Scenarios* $\langle pCES, IES \rangle$.
Let PS be a set of $Pairs$.
Let pe be an *Effect* in a previous *Cumulative Effect Scenario*.
Let sub be an *Effect Scenario* such that $sub \subset cCES$.
Let es be an *Effect Scenario* such that $es \in cWL$ and $es \subset cCES$ and $es \cup KBR \models \top$.
Let $subWorlds$ be a set of *Effect Scenarios*.
Let s be an *Effect Scenario* such that $s \in subWorlds$ and $s \subset cCES$ and $s \cup KBR \models \top$.

```
function PW(cCES, pCES)
    cWL = ∅
    for all (pe ∈ pCES) do
        sub = cCES − pe
        flag = False
        for all (es ∈ cWL) do
            if sub ⊆ es then
                flag = True
            end if
        end for
        if (flag = False) then
            if (sub ∪ KBR ⊨ ⊤) then
                cWL . ADD(sub)
            else
                subWorlds = PW(sub)
                for all (s ∈ subWorlds) do
                    cWL . ADD(s)
                end for
            end if
        end if
    end for
    return cWL
end function
```

Algorithm A.4 Pair-wise Accumulation Function

Let KBR be a consistent background knowledge base and set of rules.
Let IES be an *Immediate Effect Scenario*.
Let $pCES$ be a previous *Cumulative Effect Scenario*.
Let $cCES$ be a current *Cumulative Effect Scenario*.
Let cWL be a current *World List* of current *Cumulative Effect Scenarios*.
function ACC($pCES, IES$)
    $cWL = \emptyset$
    $cCES = pCES \cup IES$ with duplicate assertions removed
    if $(cCES \cup KBR \models \top)$ **then**
        $cWL = \{cCES\}$
    else
        $cWL =$ PW($cCES, pCES$)
    end if
    return cWL
end function

Algorithm A.5 Pair-wise Accumulation Function Extended

Let KBR be a consistent background knowledge base and set of rules.
Let iWL be an *Immediate World List*.
Let IES be an *Immediate Effect Scenario* such that $IES \in iWL$.
Let pWL be a previous *Cumulative World List*
Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let $pCES_n$ be a previous *Cumulative Effect Scenario* such that $pCES_n \in pAS$ and $pCES_n$ is the last element in pAS.

Let pH be an *Effect Scenario History* of $pCES_n$ such that $pH = pAS - pCES_n$
Let cWL be a current *Cumulative World List*.
Let cAS be an *Ancestor Sequence* such that $cAS \in cWL$ or $cAS \in PW$.
Let $cCES_n$ be a previous *Cumulative Effect Scenario* such that $cCES_n \in cAS$ and $cCES_n$ is the last element in cAS.

Let PW be a set of *Cumulative Effect Scenarios* or Possible Worlds or different $cCES_n$'s

function ACC(pAS, IES)
    $cWL = \emptyset$
    $cAS = pH$
    $cCES_n = pCES_n \cup IES$ with duplicate assertions removed
    if $(cCES \cup KBR \models \top)$ **then**
        cAS = ADD($cCES_n$)
        cWL . ADD(cAS)
    else
        PW = PW($cCES_n, pCES_n$)
        for all $(cCES_n \in PW)$ **do** ▷ cAS is equivalent to pH, a base to which $cCES_n$ can be added.
            Copy of cAS . ADD($cCES_n$)
            cWL . ADD(Copy of cAS)
        end for
    end if
    return cWL
end function

Algorithm A.6 Accumulation of $\langle T,T\rangle$, $\langle P,T\rangle$, $\langle XJ,T\rangle$, $\langle PJ,T\rangle$, $\langle IJ,T\rangle$

Let $\langle T1,T2\rangle$ be a pair of BPMN activities in sequence.
Let KBR be a consistent background knowledge base and set of rules.
Let iWL be the *Immediate World List* of $T2$.
Let IES be an *Immediate Effect Scenario* such that $IES \in iWL$.
Let pWL be the *Cumulative World List* of $T1$. Therefore pWL is the previous *Cumulative World List* of $T2$

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let $pCES_n$ be a previous *Cumulative Effect Scenario* such that $pCES_n \in pAS$ and $pCES_n$ is the last element in pAS.

Let cWL be a current *Cumulative World List* of $T2$.
Let PW be a set of *Ancestor Sequences* or Possible Worlds of $T2$
Let cAS be an *Ancestor Sequence* such that $cAS \in cWL$.
Let $cCES_n$ be a previous *Cumulative Effect Scenario* such that $cCES_n \in cAS$ and $cCES_n$ is the last element in cAS.

Let $Pair$ be a sequence $\langle pAS, IES\rangle$
Let PS be a set of $Pairs$.

function ACCUMULATE SEQUENCED ACTIVITIES(pWL, iWL)
  $cWL = \emptyset$
  $PS =$ COMBO(pWL, iWL)
  for all ($Pair \in PS$) **do**
    $PW =$ ACC(pAS, IES)
    for all ($cAS \in PW$) **do**
      cWL . ADD(cAS)
    end for
  end for
  return cWL
end function

Algorithm A.7 Accumulation of $\langle T, P\rangle$ or $\langle T, I\rangle$

Let $\langle T1, P1\rangle$ be a sequence of an activity and a parallel/inclusive gateway split.
Let pWL be the *Cumulative World List* of $T1$. Therefore pWL is the previous *Cumulative World List* of $P1$

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let $pCES_n$ be a previous *Cumulative Effect Scenario* such that $pCES_n \in pAS$ and $pCES_n$ is the last element in pAS.

Let cWL be a current *Cumulative World List* of $P1$.
Let cAS be an *Ancestor Sequence* such that $cAS \in cWL$.

function ACCUMULATE $\langle T, P\rangle(pWL)$
  $cWL = \emptyset$
  for all $(pAS \in pWL)$ **do**
    $cAS = pAS$. ADD$(pCES_n)$
    cWL . ADD(cAS)
  end for
  return cWL
end function

Algorithm A.8 Accumulation of $\langle T, X\rangle$, $\langle XJ, X\rangle$, $\langle PJ, X\rangle$, $\langle IJ, X\rangle$, $\langle P, X\rangle$

Let $\langle T1, X1\rangle$ be a sequence of an activity and an exclusive gateway split.
Let pWL be the *Cumulative World List* of $T1$. Therefore pWL is the previous *Cumulative World List* of $X1$

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let cWL be a current *Cumulative World List* of $X1$.

function ACCUMULATE $\langle T, X\rangle(pWL)$
  $cWL = \emptyset$
  for all $(pAS \in pWL)$ **do**
    cWL . ADD(pAS)
  end for
  return cWL
end function

Algorithm A.9 Accumulation of $\langle X, T\rangle$ or $\langle I, T\rangle$

Let $\langle X1, T1\rangle$ be a sequence of an exclusive gateway split and an activity.
Let KBR be a consistent background knowledge base and set of rules.
Let CS be a *Condition Scenario* annotated to the sequence flow between $X1$ and $T1$
Let iWL be the *Immediate World List* of $T1$.
Let IES be an *Immediate Effect Scenario* such that $IES \in iWL$.
Let xWL be the *Cumulative World List* of $X1$. Therefore xWL is the previous *Cumulative World List* prior to being filtered with CS

Let xAS be an *Ancestor Sequence* such that $xAS \in xWL$.
Let $xCES_n$ be a previous *Cumulative Effect Scenario* such that $xCES_n \in xAS$ and $xCES_n$ is the last element in xAS.

Let pWL be the previous *Cumulative World List* of $T1$ such that $pWL \subset xWL$

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let $pCES_n$ be a previous *Cumulative Effect Scenario* such that $pCES_n \in pAS$ and $pCES_n$ is the last element in pAS.

Let cWL be a current *Cumulative World List* of $T1$.
Let PW be a set of *Ancestor Sequences* or Possible Worlds of $T1$
Let cAS be an *Ancestor Sequence* such that $cAS \in cWL$.
Let $cCES_n$ be a previous *Cumulative Effect Scenario* such that $cCES_n \in cAS$ and $cCES_n$ is the last element in cAS.

Let $Pair$ be a sequence $\langle pAS, IES\rangle$
Let PS be a set of $Pairs$.

function ACCUMULATE $\langle X, T\rangle(xWL, iWL)$
  $cWL = \emptyset$
  $pWL =$ DEC(xWL, CS)
  $PS =$ COMBO(pWL, iWL)
  for all $(Pair \in PS)$ **do**
    $PW =$ ACC(pAS, IES)
    for all $(cAS \in PW)$ **do**
      cWL . ADD(cAS) ▷ Copy each cAS into cWL.
    end for
  end for
  return cWL
end function

Algorithm A.10 Accumulation of $\langle X, X\rangle$ or $\langle I, X\rangle$

Let $\langle X1, X2\rangle$ be a sequence of two exclusive gateway splits.
Let CS be a *Condition Scenario* annotated to the sequence flow between $X1$ and $X2$
Let xWL be the *Cumulative World List* of $X1$. Therefore xWL is the previous *Cumulative World List* prior to being filtered with CS

Let pWL be the previous *Cumulative World List* of $T1$ such that $pWL \subset xWL$

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let cWL be a current *Cumulative World List* of $X2$.

function ACCUMULATE $\langle X, X\rangle(xWL)$
  $cWL = \emptyset$
  $pWL = \text{DEC}(xWL, CS)$
  for all $(pAS \in pWL)$ **do**
    cWL . ADD(pAS) ▷ Copy each pAS into cWL.
  end for
  return cWL
end function

Algorithm A.11 Accumulation of $\langle X, P\rangle$ or $\langle X, I\rangle$

Let $\langle X1, P1\rangle$ be a sequence of an exclusive gateway split and a parallel gateway split.
Let CS be a *Condition Scenario* annotated to the sequence flow between $X1$ and $P1$
Let xWL be the *Cumulative World List* of $X1$. Therefore xWL is the previous *Cumulative World List* prior to being filtered with CS

Let pWL be the previous *Cumulative World List* of $P1$ such that $pWL \subset xWL$

Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let $pCES_n$ be a previous *Cumulative Effect Scenario* such that $pCES_n \in pAS$ and $pCES_n$ is the last element in pAS.

Let cWL be a current *Cumulative World List* of $P1$.
Let cAS be an *Ancestor Sequence* such that $cAS \in cWL$.

function ACCUMULATE $\langle X, P\rangle(xWL)$
  $cWL = \emptyset$
  $pWL = \text{DEC}(xWL, CS)$
  for all $(pAS \in pWL)$ **do**
    $cAS = pAS$. ADD$(pCES_n)$ ▷ Duplicate $pCES_n$ in each pAS.
    cWL . ADD(cAS) ▷ Copy each cAS into cWL.
  end for
  return cWL
end function

Algorithm A.12 Accumulation of $\langle B, XJ \rangle$

Let $\langle B, XJ1 \rangle$ be a sequence of a set of incoming branches and an exclusive gateway join.
Let BWL be a set of *World Lists* from the incoming branches to $XJ1$.
Let pWL be a previous *Cumulative World List* of $XJ1$ such that $pWL \in BWL$.
Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let cWL be a current *Cumulative World List* of $XJ1$.

```
function ACCUMULATE ⟨B, XJ⟩(BWL)
    cWL = ∅
    for all (pWL ∈ BWL) do
        for all (pAS ∈ pWL) do
            cWL . ADD(pAS)
        end for
    end for
    return cWL
end function
```

Algorithm A.13 Accumulation of $\langle B, PJ\rangle$, $\langle B, IJ\rangle$

Let $\langle B, PJ1\rangle$ be a sequence of a set of incoming branches and a parallel gateway join.

Let KBR be a consistent background knowledge base and set of rules.
Let BWL be a set of *World Lists* from the incoming branches to $PJ1$.
Let pWL be a previous *Cumulative World List* of $PJ1$ such that $pWL \in BWL$.
Let SC be a set of *Branch Clusters*.
Let BSC be a set of SC
Let BG be a *Branch Group*
Let SBG be a set of *Branch Groups*
Let BC be a *Branch Combination*
Let SBC be a set of *Branch Combinations*
Let WL be a *Cumulative World List*.
Let cWL be a current *Cumulative World List* of $P1$.

```
function ACCUMULATE ⟨B, PJ⟩(BWL)
    cWL = ∅
    for all (pWL ∈ BWL) do
        SC = CLUSTER(pWL)                          ▷ (See AlgA.14)
        BSC . ADD(SC)
    end for
    SBG = ∅
    SBG = GROUP(BSC)                               ▷ (See AlgA.15)
    SBC = ∅
    for all (BG ∈ SBG) do
        sbc = ∅
        sbc = BRANCHCOMBO(BG)                      ▷ (See AlgA.17)
        for all (BC ∈ sbc) do
            SBC . ADD(BC)
        end for
    end for
    for all (BC ∈ SBC) do
        WL = ∅
        WL = JOINACC(BC)                           ▷ (See AlgA.16)
        cWL = ADD(WL)
    end for
    return cWL
end function
```

Algorithm A.14 Cluster Function

Let $\langle B, PJ1 \rangle$ be a sequence of a set of incoming branches and a parallel gateway join.
Let BWL be a set of *World Lists* from the incoming branches to $PJ1$.
Let pWL be a previous *Cumulative World List* of $PJ1$ such that $pWL \in BWL$.
Let pAS be an *Ancestor Sequence* such that $pAS \in pWL$.
Let $pCES_n$ be a previous *Cumulative Effect Scenario* such that $pCES_n \in pAS$ and $pCES_n$ is the last element in pAS.

Let pAS_h be an *Effect Scenario History* such that $pAS_h = pAS - pCES_n$.
Let C be a *Cluster* such that $C \subseteq pWL$
Let cAS be an *Ancestor Sequence* such that $cAS \in C$
Let $cCES_n$ be a previous *Cumulative Effect Scenario* such that $cCES_n \in cAS$ and $cCES_n$ is the last element in cAS.

Let cAS_h be an *Effect Scenario History* such that $cAS_h = cAS - cCES_n$.
Let SC be a set of *Clusters* such that $\forall C_i \in SC(\sum_{i=1}^{n} |C_i| = |pWL|)$

function CLUSTER(pWL)
  $SC = \emptyset$
  while ($|pWL| > 0$) **do**
    $C = \emptyset$
    for all ($pAS \in pWL$) **do**
      if ($C = \emptyset$) **then**
        C . ADD(pAS)
        $pWL = pWL - pAS$
      else
        if ($pAS_h = cAS_h$) **then** ▷ Only needs to check one cAS in C
          C . ADD(pAS)
          $pWL = pWL - pAS$
        end if
      end if
    end for
    if ($C \neq \emptyset$) **then**
      SC . ADD(C)
    end if
  end while
  return SC
end function

Algorithm A.15 Branch Group Function

Let B be a set of incoming branches to a gateway join.
Let $|B|$ be the number of incoming branches to a gateway join.
Let C be a *Cluster* of *Ancestor Sequences*, with the same origin, from a single branch.
Let AS be an *Ancestor Sequence* such that $AS \in C$.
Let CES_n be a *Cumulative Effect Scenario* such that $CES_n \in AS$ and CES_n is the last element in AS.
Let AS_h be an *Effect Scenario History* such that $AS_h = AS - CES_n$.
Let SC be a set of C.
Let $tempSC$ be a temporary set of C.
Let BSC be a set of SC such that $|BSC| = |B|$.
Let $tempBSC$ be a temporary set of SC.
Let G be a *Branch Group* such that $|G| \leq |B|$.
Let BG be a set of G.

```
function branchGroup(BSC)
    BG = ∅
    tempBSC = copy(BSC)
    while (|BSC| > 0) do
        G = ∅
        tempBSC = ∅
        for all (SC ∈ BSC) do
            if (G = ∅) then
                G . add(C)
                SC = SC − C
            else
                tempSC = ∅
                matchFound = False
                for all (C ∈ SC) do
                    if (C.AS_h = G.C.AS_h) then          ▷ Compare Histories
                        G . add(C)
                        matchFound = True
                    else
                        tempSC . add(C)
                    end if
                end for
                if (matchFound = True) then
                    SC = tempSC                          ▷ SC with matched C removed.
                end if
            end if
            if (SC ≠ ∅) then
                tempBSC = add(SC)
            end if
        end for
        BSC = tempBSC
        BG . add(G)
    end while
    return BG
end function
```

Algorithm A.16 Join Accumulation Function

Let $\langle B, PJ1\rangle$ be a sequence of a set of incoming branches and a parallel gateway join.
Let KBR be a consistent background knowledge base and set of rules.
Let $SBCS$ be a set of *Branch Combinations*.
Let BC be a *Branch Combination* such that $BC \in SBC$.
Let AS be an *Ancestor Sequence* such that $AS \in BC$ or $AS \in cWL$
Let cWL be a current *Cumulative World List* of $PJ1$.
Let CES_n be a *Cumulative Effect Scenario* such that $CES_n \in AS$ and CES_n is the last element in AS.

Let CES_{n-1} be a *Cumulative Effect Scenario* such that CES_{n-1} is the second last element in AS.

Let $uSet$ be a set of Cumulative Effect Scenarios such that $\forall CES_n \in AS \in BC(CES_n \in uSet)$.

Let $uCES$ be a *Cumulative Effect Scenario* that is the union of all *Effects* in all *Cumulative Effect Scenarios* in $uSet$.

function JOINACC(SBC)
  for all ($BC \in SBC$) **do**
    $uSet = \emptyset$
    for all ($AS \in BC$) **do**
      $uSet =$ ADD(CES_n)
    end for
    $uCES = \emptyset$
    for all ($CES_n \in unionSet$) **do**
      for all ($e \in CES_n$) **do**
        $uCES$. ADD(e)
      end for
    end for
    $uCES = uCES - CES_{n-1}$ ▷ CES_{n-1} will be identical for all $AS \in BC$
    if ($uCES \cup KBR \models \perp$) **then**
      Alert User of erroneous BPMN design
    else
      $cCES =$ ACC($CES_{n-1}, union$)
      $AS = AS - CES_{n-1} - CES_n$
      AS . ADD($cCES$)
      cWL . ADD(AS)
    end if
  end for
  return cWL
end function

Algorithm A.17 Branch Combinatorial Function

Let BG be a Branch Group.
Let C be a Cluster such that $C \in BG$.
Let *temp* be a temporary set of *Branch Combinations*
Let BC be a Branch Combination.
Let AS be an Ancestor Sequence such that $AS \in C$ or $AS \in BC$
Let SBC be a set of *Branch Combinations* such that $BC \in SBC$.

```
function branchCombo(BG)
    SBC = ∅
    for all (C ∈ BG) do
        if (SBC = ∅) then
            for all (AS ∈ C) do
                CreateNewBC
                BC . add(AS)
                SBC . add(BC)
            end for
        else
            temp = ∅
            for all (BC ∈ SBC) do
                for all (AS ∈ C) do
                    BCcopy = copy(BC)
                    BCcopy . add(AS)
                    temp . add(BCcopy)
                end for
            end for
            for all (BC ∈ temp) do
                SBC . add(BC)
            end for
        end if
    end for
    return SBC
end function
```

Appendix B

Standards and Specifications

Business Process Model and Notation (BPMN)
http://www.omg.org/spec/BPMN/2.0.2/
http://www.omg.org/spec/BPMN/1.2/

Resource Description Framework (RDF)
https://www.w3.org/TR/REC-rdf-syntax/

Web Ontology Language (OWL)
https://www.w3.org/TR/owl-features/

Extensible Markup Language (XML)
https://www.w3.org/TR/2006/REC-xml11-20060816/

XML Metadata Interchange (XMI)
http://www.omg.org/spec/XMI/

Web Service Definition Language (WSDL)
https://www.w3.org/TR/wsdl

Semantic Annotations for Web Service Definition Language (SAWSDL)
https://www.w3.org/2002/ws/sawsdl/

Web Service Semantics (WSDL-S)
https://www.w3.org/Submission/WSDL-S/

Web Services Modelling Ontology (WSMO)
https://www.w3.org/Submission/WSMO/

Appendix C

Governing Organisations

Object Management Group (OMG)
`http://www.omg.org/`

World Wide Web Consortium (W3C)
`https://www.w3.org/`

Web Services Modelling Ontology (WSMO)
`http://www.wsmo.org/`

Appendix D

Applications

Eclipse
http://www.eclipse.org

Eclipse Galileo
http://www.eclipse.org/galileo/projects.php

Eclipse STP modeler
https://java.net/projects/bpmn-modeler/sources/source-code-reposito
content/org.eclipse.stp.bpmn.feature/trunk/feature.xml?rev=14